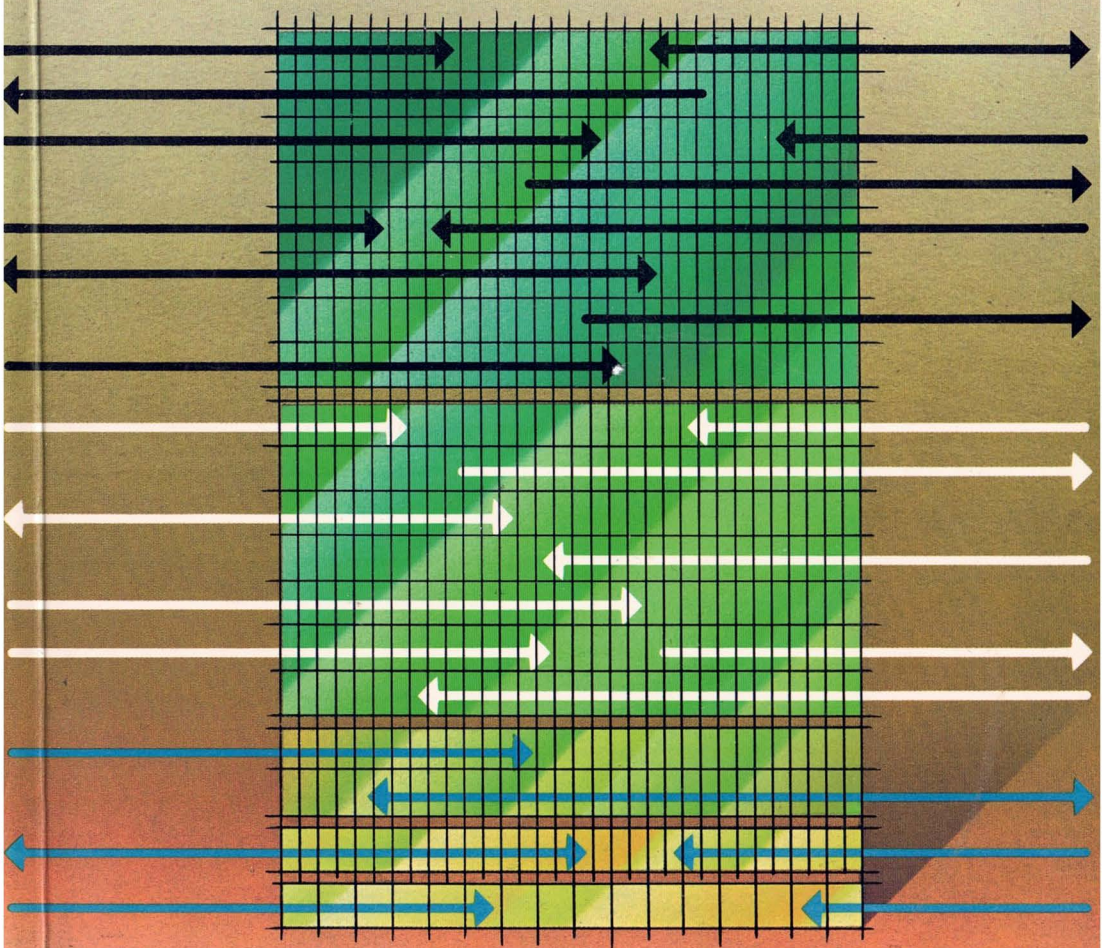
Elektronik.
Wir bauen die Elemente.

VALVO



Der 16bit-Mikroprozessor SC 68000

Befehlsvorrat

DER 16BIT-MIKROPROZESSOR SC 68000

VALVO

BEFEHLSVORRAT

Der 16bit-Mikroprozessor SC 68000

Befehlsvorrat

bearbeitet von J. Koch

VALVO

Diese Broschüre enthält den Befehlsvorrat des Mikroprozessors SC 68000 sowie alle Angaben, die zu seiner Programmierung erforderlich sind. Eine ausführliche Hardware-Beschreibung befindet sich in der im November 1982 erschienenen Valvo Broschüre „Der Mikroprozessor SC 68000 — Eigenschaften" (ISBN 3-87095-255-5, Verlag Boysen + Maasch, Hamburg) sowie im gültigen Datenblatt.

Herrn Rainer Kind danke ich für zahlreiche Hinweise und anregende Diskussionen zu beiden Broschüren.

J. Koch

1. Ausgabe April 1983

Herausgeber: Valvo Unternehmensbereich Bauelemente der Philips GmbH, Burchardstraße 19, 2000 Hamburg 1

Diese Broschüre gibt keine Auskunft über Liefermöglichkeiten. Die angegebenen Daten dienen allein der Produktbeschreibung und sind nicht als zugesicherte Eigenschaften im Rechtssinne aufzufassen. Etwaige Schadensersatzansprüche gegen uns — gleich aus welchem Rechtsgrund — sind ausgeschlossen, soweit uns nicht Vorsatz oder grobe Fahrlässigkeit trifft.
Es wird keine Gewähr übernommen, daß die angegebenen Schaltungen oder Verfahren frei von Schutzrechten Dritter sind.

Ein Nachdruck — auch auszugsweise — ist nur zulässig mit Zustimmung des Herausgebers und mit genauer Quellenangabe.

Verlag Boysen + Maasch, 2000 Hamburg 1, Hermannstr. 31
ISBN 3-87095-257-1
Satz und Druck: A. Sutter Druckerei GmbH, 4300 Essen 1
Einbandentwurf: Holger Börnsen, 2000 Hamburg

Inhalt

1. Einleitung: Innovation durch Integration 7

2. Register und Datenorganisation 14

2.1. Operandenformate 14
2.2. Registerformate . 14
2.2.1. Datenregister 14
2.2.2. Adreßregister 14
2.2.3. Statusregister . . . 16
2.3. Datenorganisation im Speicher . 17
2.4. Stacks und Queues . 17
2.4.1. System-Stacks 19
2.4.2. Software-Stacks 19
2.4.3. Queues 21

3. Befehlsformate und Adressierungsarten 23

3.1. Bestimmung der Operandenadresse . . . 23
3.2. Programm- und Datenbereichszuweisungen 26
3.3. Effektive Adressen 26
3.3.1. Adressierungsarten Register direkt . 27
3.3.2. Adressierungsarten Register indirekt 27
3.3.3. Adressierungsarten Daten absolut 31
3.3.4. Adressierungsarten Relativ zum Programmzähler 32
3.3.5. Adressierungsarten Daten unmittelbar 34
3.4. Bedingungskodes und Statusregister 35
3.5. Impliziter Bezug auf Register 35

4. Ausnahmeverarbeitung und -vektoren 35

4.1. Ausnahmearten . . . 35
4.2. Ausnahmeverarbeitung 37
4.3. Ausnahmevektoren . 37

5. Überblick über den Befehlsvorrat 39

6. Beschreibung des Befehlsvorrats 50

6.1. Erläuterungen zur Befehlsbeschreibung 50
6.2. Abkürzungen und Definitionen bei der Befehlsbeschreibung 51
6.3. Befehlsbeschreibung in alphabetischer Reihenfolge 53

7. Befehlsvorrat-Zusammenfassung 172

8. Berechnung der Bedingungskodes 177

8.1. Allgemeines 177
8.2. Bedingungskoderegister 178
8.3. Tabelle zur Berechnung des Bedingungskodes 178

9. Mikroprogramm-Architektur des SC 68000 181

10. Prefetch-Mechanismus 182

10.1. Allgemeines . . 182
10.2. Befehls-Prefetch 183
10.3. Daten-Prefetch . 185

11. Programmbeispiele 186

1. Einleitung: Innovation durch Integration

Der 16bit-Mikroprozessor SC 68000 gehört bezüglich seiner Konzeption zu einer neuen Rechnergeneration, die neben einer hohen Verarbeitungsgeschwindigkeit und großen Adressierräumen auch Aspekte moderner Programmierverfahren berücksichtigt.

Bevor in dem vorliegenden Buch auf den Befehlsvorrat dieser MPU näher eingegangen wird, sollen zuerst die grundlegenden Systemgedanken etwas näher erläutert werden.

Architektur-Merkmale

Der SC 68000 ist in seiner internen Registerstruktur auf 32bit-Wortbreite ausgelegt — sowohl für Datenregister als auch für Adreßregister. Er bildet die Basis für zukünftig verfügbare MPU-Versionen und erlaubt so die Mitnahme vorhandener 68000-Software auf diese. Bei Arbeiten mit Langwortoperanden (32bit-Wortbreite) wird durch die interne 32bit-Struktur eine kürzestmögliche Verarbeitungszeit erreicht.

Der 23bit-Adreßbus erlaubt die Adressierung von 8 Megawörtern, entsprechend einem Adressierraum von 16 Megabytes. Dadurch, daß keinerlei Segmentierung innerhalb des gesamten Adreßbereiches auftritt, lassen sich große Datenfelder bearbeiten — z. B. wie dies zur Bildpunktverarbeitung in Grafik-Displays erforderlich ist. Die Adreßregister selbst verfügen über eine eigene ALU, wodurch eine außerordentlich schnelle Adreßberechnung bei Verwendung der verschiedenen Adressierungsarten erreicht wird.

An seiner externen Schnittstelle sind der 16bit-Datenbus und der 23bit-Adreßbus separat herausgeführt. Dies führt zu sehr kurzen Zykluszeiten, da — anders als bei Bussystemen im Multiplex-Betrieb — keine Umschaltverluste im zeitlichen Ablauf eintreten. Der Datenbus arbeitet vollkommen asynchron. Über Datenstrobe-Signale von der MPU und ein Quittungssignal von den adressierten Peripherie- oder Speicherschaltungen wird ein sicherer Quittungsbetrieb erreicht. Auf diese Weise können auch langsame Peripherieschaltungen eingesetzt werden, ohne daß der Datenaustausch mit schnelleren Schaltungen dadurch beeinflußt wird. Speziell für das 68000-System zugeschnittene Peripherieschaltungen (z. B. der SC 68681 Double UART) — Übersicht s. Bild 1 — erzeugen dieses Quittungssignal von sich aus. Der SC 68000 erlaubt zusätzlich den Anschluß von synchronen Peripherieschaltungen ohne Quittungssignal und stellt hierfür drei besondere Leitungen zur Verfügung.

Eine Besonderheit im Datenverkehr mit dem Speicher bildet ein nicht unterbrechbarer Lese/Schreibbuszyklus. Dieser Zyklus erlaubt das Lesen, Testen und Schreiben von Speicherbits, ohne daß (z. B. in Multiprozessorsystemen) eine andere MPU diesen

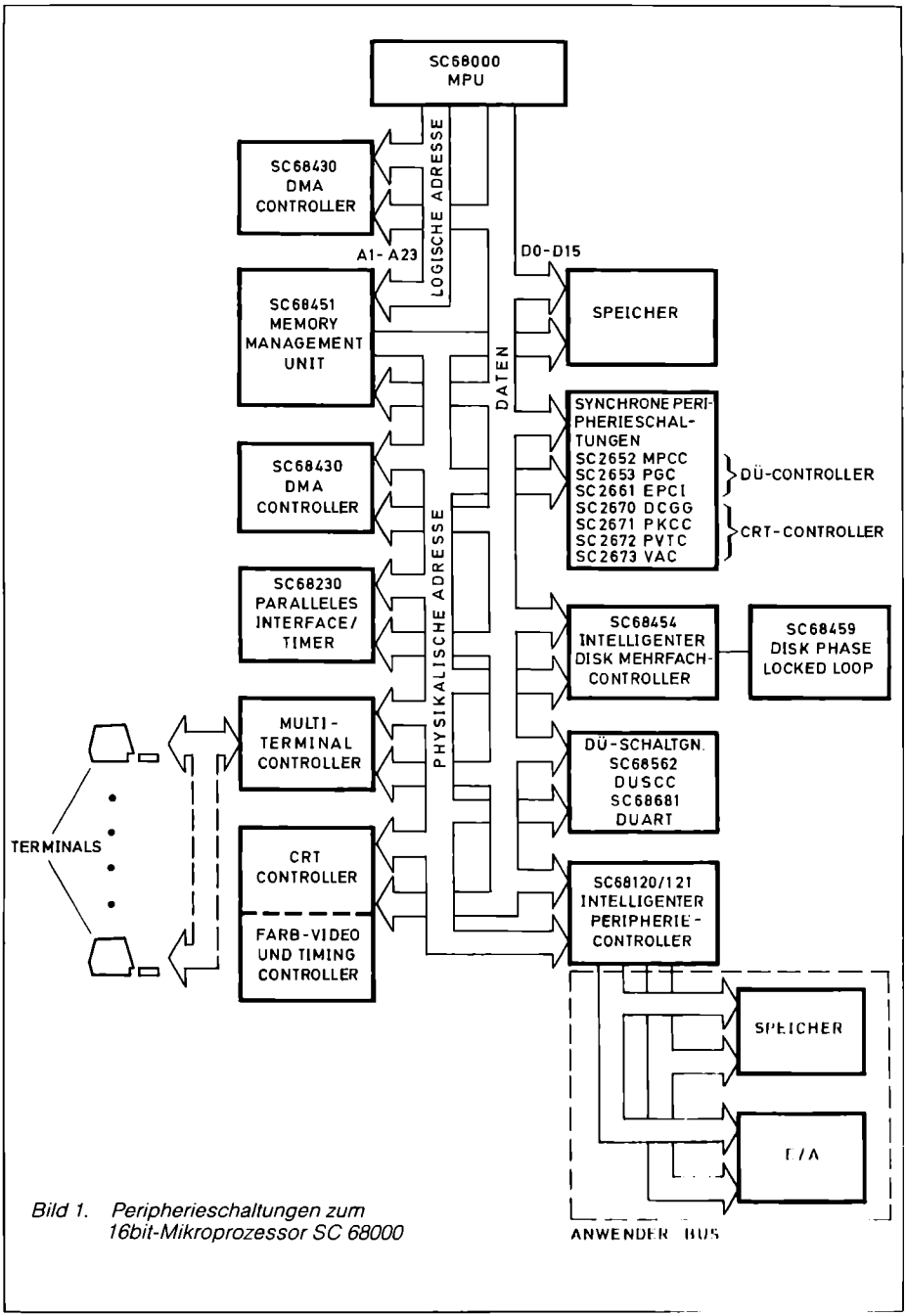


Bild 1. Peripherieschaltungen zum 16bit-Mikroprozessor SC 68000

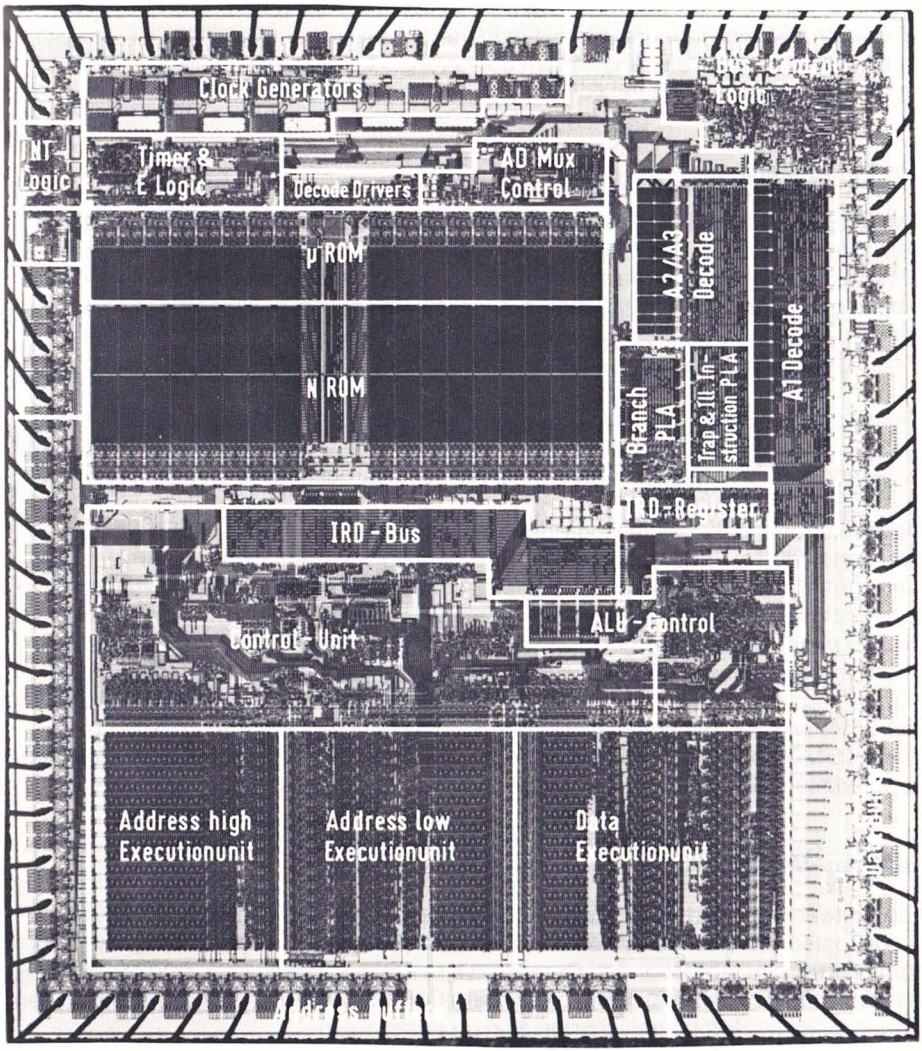


Bild 2. Kristallbild des Mikroprozessors SC 68000

Vorgang unterbrechen kann — eine unabdingbare Voraussetzung, um die Statusbits (Semaphoren) von gemeinsamen Datenfeldern bearbeiten zu können.

Bezüglich der verwendbaren Datentypen ermöglicht es der SC 68000, mit Bits, Bytes, 16bit-Wörtern und 32bit-Langwörtern operieren zu können. Einige spezielle Befehle sehen auch die Verwendung von binär kodierten Dezimalziffern (BCD-Digits) vor.

Hardware-Merkmale

Der SC 68000 ist ein mikroprogrammkodierter Prozessor. Jeder Anwender-Befehl wird intern in Mikroschritte zerlegt — diese wiederum in noch kleinere Nanoschritte. Das Mikro- und Nanoprogramm selbst ist in einem maskenprogrammierten ROM abgelegt, wobei 20% der verfügbaren Kapazität für spätere Erweiterungen nicht belegt sind. Bild 2 zeigt die außerordentlich reguläre Schaltungsstruktur, wie sie auf einer Kristallfläche von 44 mm^2 verwirklicht wurde.

Dem Anwender stehen — aufgerufen durch von außen anlegbare Adressen — insgesamt 192 Interruptvektoren zur Verfügung sowie zusätzlich 7 Autovektoren bei Verwendung von Peripherieschaltungen mit synchron arbeitender Schnittstelle.

Jede Interruptmeldung kann individuell mit einer von 7 Prioritäten belegt werden, von denen die höchste nicht maskierbar ist. Hervorzuheben ist an dieser Stelle die ungewöhnliche Vielzahl von internen Interruptreaktionen, sobald es zu irregulären Zuständen innerhalb des Systemes kommt. So führen z. B. eine Division durch Null, die Verwendung unzulässiger Befehlskodes oder die Überschreitung vorgegebener Wertgrenzen in einem Datenregister zu Interrupts (Traps) mit definierten Vektoren.

Zur Erhöhung der Systemgeschwindigkeit enthält der SC 68000 ein Zweiwort-Prefetch-Mechanismus. Dadurch wird erreicht, daß schon während des internen Mikroprogrammablaufes der nächste Befehl aus dem Programmspeicher gelesen wird.

Software-Merkmale

Die Software erfordert einen zunehmend größeren Anteil an den Gesamtentwicklungskosten von Rechnersystemen, so daß der Einsatz von höheren Programmiersprachen und entsprechenden Compilern eine Notwendigkeit ist. Der SC 68000 stellt spezielle Befehle zur Verfügung, die es dem Compiler erlauben, effiziente Maschinenkodes für kurze Programmlaufzeiten zu generieren.

Einige Beispiele:

LNK (Link and Allocate)
UNLK (Unlink)

Diese Befehle erlauben es, einer Task (Aufgabe) den erforderlichen Arbeitsspeicher im Stack zuweisen zu können bzw. diesen nach Beendigung der Task wieder abzugeben.

MOVEM (Move Multiple Registers)

Mehrere oder alle internen Register können mit einem Befehl in den Speicher gerettet werden, wodurch eine außerordentlich kurze Reaktionszeit bei Interruptanforderungen erreicht wird.

LEA (Load Effective Address)
PEA (Push Effective Address)

Diese Befehle dienen zur Berechnung von Adressen aus Parametern und für Adreßfelder mit mehreren Indizes — z. B. zur Darstellung dreidimensionaler Kennfelder.

S_{CC} (Set Conditionally)

Mit diesem Befehl läßt sich eine von 16 unterschiedlichen Bedingungen im Statusregister testen, z. B. Überlaufbit gesetzt — nicht gesetzt, Ergebnis gleich oder größer — gleich oder kleiner, kleiner — gleich oder minus oder wahr oder unwahr, . . .).

Alle o. a. Beispiele ergeben in Verbindung mit den vielfältigen Adressierungsarten eine gute Unterstützung von modularer Software sowie von positionsunabhängigen und wiedereintrittsfähigen Programmkodes. In einigen nachfolgend aufgeführten Beispielen zeigt sich die Ähnlichkeit von PASCAL-Prozeduren und SC 68000-Befehlen:

PASCAL-Merkmale	**SC 68000-Merkmale**
Procedure Calls	Mehrere Adreßregister erlauben die Verwendung verschiedener Stack pointer für verschachtelte Programmaufrufe.
Push Parameters and Adresses	Die Befehle MOVE und PEA laden Daten bzw. berechnete Adressen in den Stack.
Call Procedure	Sprung in eine Subroutine.
Return from Procedure	Rücksprung aus einer Subroutine.
Establish Local Environment	Der Befehl LINK rettet ein Adreßregister (Frame Pointer) in den Stack. Der neue Stand des Stack pointers wird in das Adreßregister gebracht und mit einer vorgegebenen Adreßdistanz versehen.
Reestablish Environment	Der Befehl UNLK kehrt die LINK-Prozedur um.
Save Register Subset	Der Befehl MOVEM rettet mehrere Register in den Speicher.
Reload Register Subset	Der Befehl MOVEM lädt mehrere Register mit dem Speicherinhalt.
Restore Stack	Daten können unmittelbar in den Stack pointer addiert werden.

Neben höheren Programmiersprachen unterstützt der SC 68000 auch moderne Betriebssysteme. So wird z. B. während des Programmablaufes zwischen den Betriebsarten „Supervisor" und „Anwender" unterschieden. In der Betriebsart „Anwender" sind einige Befehle nicht zulässig und führen automatisch zu einem internen Interrupt mit entsprechendem Vektor. Über externe Leitungen zeigt der SC 68000 an, in welchem der beiden Betriebsarten er sich bei jedem Zyklus befindet. Alle Interrupts führen automatisch in den Zustand „Supervisor". Auf diese Weise lassen sich durch unbeabsichtigte, fehlerhafte Anwendersoftware hervorgerufene Änderungen in der Betriebssystemsoftware oder im Statusregister vermeiden. Eine besondere Betriebsart „Trace" erlaubt den schrittweisen Ablauf und Test einer Anwendersoftware unter Kontrolle des Betriebssystemes.

Weitere MPU-Entwicklungen

Der SC 68000 ist Grundlage für eine Familie von Mikroprozessoren, die in ihrer Leistungsfähigkeit verschiedenen Anforderungen gerecht werden (Bild 3). Der SC 68008 hat eine zum SC 68000 identische Architektur, verwendet jedoch einen 8bit-Datenbus und 20bit-Adreßbus. Die interne Wortbreite ist weiterhin 32 bit. Im Gegensatz zum SC 68000 benötigt der SC 68008 nur ein 48-Pin-DIL-Gehäuse.

Der SC 68010 ist sowohl im Interface als auch im Befehlssatz kompatibel zum SC 68000, erlaubt jedoch durch Einbringen weiterer Register und einer zusätzlichen Statusrettung die Realisierung virtueller Speicherkonzepte.

Der SC 68020 ist ein echter 32bit-Mikroprozessor, der in seiner Leistungsfähigkeit der von Mainframe-Computern entspricht. Auch bei der Entwicklung des SC 68020 wurde auf Software-Kompatibilität zum SC 68000 geachtet. Der Befehlssatz ist jedoch wesentlich erweitert und schließt Gleitkomma-Arithmetik ein. Betriebssystem- und Compiler-Software werden verstärkt unterstützt. Dazu zählen z. B. Coprozessor-Operationen sowie die Bearbeitung von ASCII-Zeichen und Bit-Feldern. Die Programmlaufzeiten des SC 68020 sind gegenüber denen des SC 68000 stark verkürzt. Neben der Verfügbarkeit von zusätzlichen Befehlen wird dies durch Implementierung eines Befehl-Caches und erhöhte Busgeschwindigkeit erreicht. Ebenso lassen sich Konzepte mit Cache-Speichern, die unmittelbar an der MPU angeordnet sind, realisieren.

Weitere Informationen, die für die folgenden Kapitel von Bedeutung sind, enthält das in derselben Reihe erschienene Valvo Buch: Der 16bit-Mikroprozessor SC 68000 — Eigenschaften (Ausgabe November 1982).

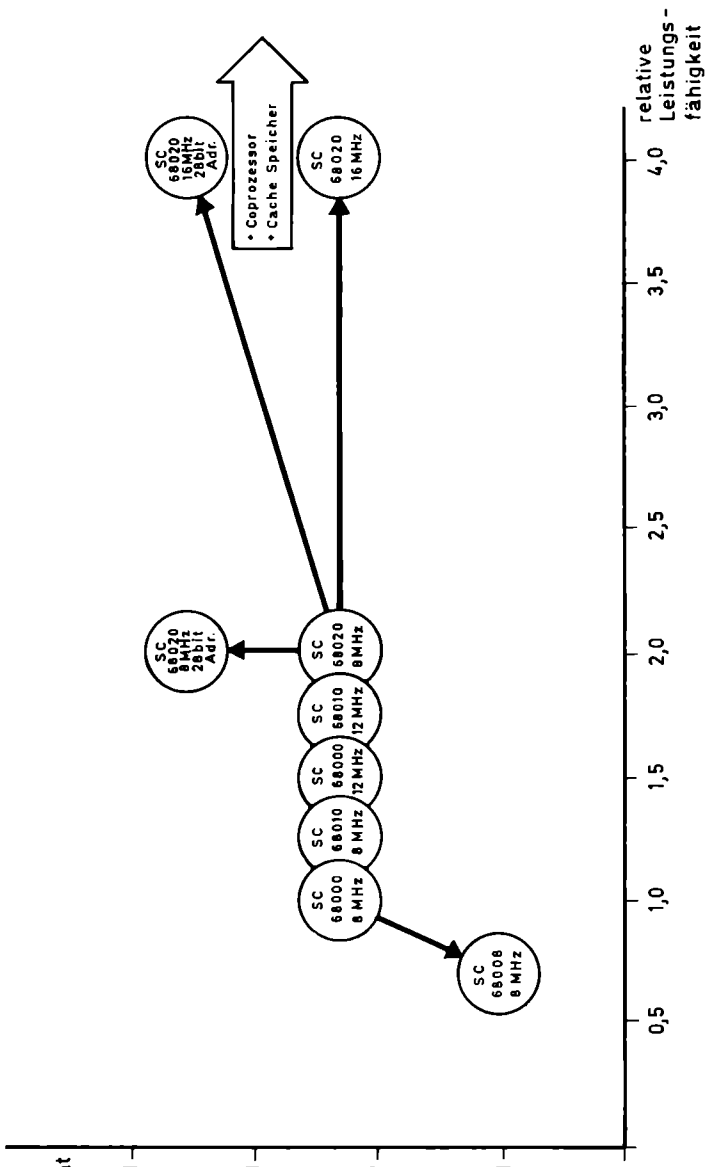


Bild 3. Leistungsfähigkeit und Adreßkapazität weiterer Mikroprozessoren der Familie 68000

2. Register und Datenorganisation

2.1. Operandenformate

Die Operanden der Befehle des Mikroprozessors SC 68000 können definitionsgemäß folgende Formate haben:

1 Byte ≙ 8 Bits
1 Wort ≙ 16 Bits
1 Langwort ≙ 32 Bits

Für jeden Befehl wird das Operandenformat entweder explizit im Befehl vorgegeben, oder es ist implizit im Befehl enthalten. Alle Befehle mit expliziter Angabe können Byte-, Wort- oder Langwortoperanden bearbeiten. Befehle mit impliziter Vorgabe unterstützen die drei Operandenlängen individuell.

2.2. Registerformate

Die Registerformate des Mikroprozessors SC 68000 sind den obengenannten Formaten angepaßt. Wie Bild 4 zeigt, enthält dieser Prozessor siebzehn 32bit-Register, einen 32bit-Programmzähler (PC) und ein 16bit-Statusregister. Die ersten 8 Register (D0—D7) sind als Datenregister für Bit- (1 bit), Byte- (8 bit), Wort- (16 bit) und Langwortoperationen (32 bit) vorgesehen. Der zweite Satz von 9 Registern (A0—A7, A7′) dient als Adreßregister und Stack pointer. Außerdem sind diese Register für Wort- und Langwort-Adreßoperationen verwendbar. Alle 17 Register lassen sich als Indexregister verwenden.

2.2.1. Datenregister

Wie bereits ausgeführt, kann jedes der 8 Datenregister 32 Bits aufnehmen. Byte-Operanden belegen die 8 unteren Bits, Wortoperanden die unteren 16 Bits und Langwortoperanden die gesamten 32 Bits. Das niedrigstwertige Bit wird mit 0, das höchstwertige Bit mit 31 bezeichnet.

Wird das Datenregister als Quellen- oder Zieloperand verwendet und beträgt das Operandenformat nicht 32 Bits, so wird nur der jeweils untere Teil geändert; der verbleibende obere Teil wird weder gebraucht noch verändert.

2.2.2. Adreßregister

Jedes der Adreßregister sowie der Stack pointer besteht aus 32 Bits und kann eine volle 32bit-Adresse enthalten. Die Adreßregister arbeiten nicht mit byteweise orientierten Operanden. Daher wird, wenn ein Adreßregister als Quellenoperand dient, ent-

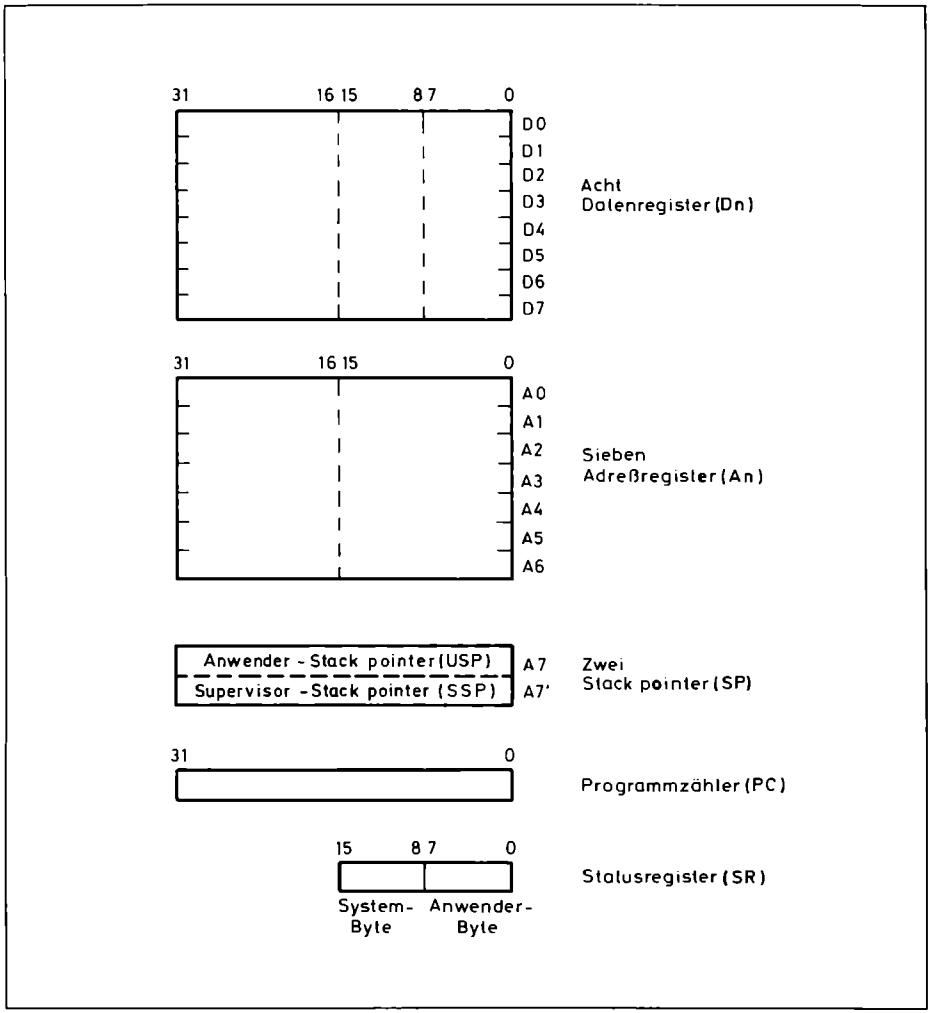


Bild 4. Registersatz des Mikroprozessors SC 68000

weder das untere Wort oder das gesamte Langwort als Operand verwendet, was vom Operandenformat abhängt. Wird ein Adreßregister als Zieloperand eingesetzt, so ist das gesamte Register davon betroffen, unabhängig vom Operandenformat. Beträgt dieses ein Wort, so werden alle anderen Operanden vorzeichenbehaftet auf 32 Bits erweitert, bevor die Operation durchgeführt wird.

2.2.3. Statusregister

Das Statusregister (Bild 5) enthält sowohl die Interrupt-Maske (7 Ebenen vorhanden) als auch die Bedingungsbits Erweiterung (X), Negativ (N), Null (Z), Überlauf (V) und Übertrag (C). Zusätzliche Statusbits zeigen an, ob der Prozessor in der Betriebsart Trace (T) arbeitet und ob er sich im Supervisor-Zustand (S) befindet.

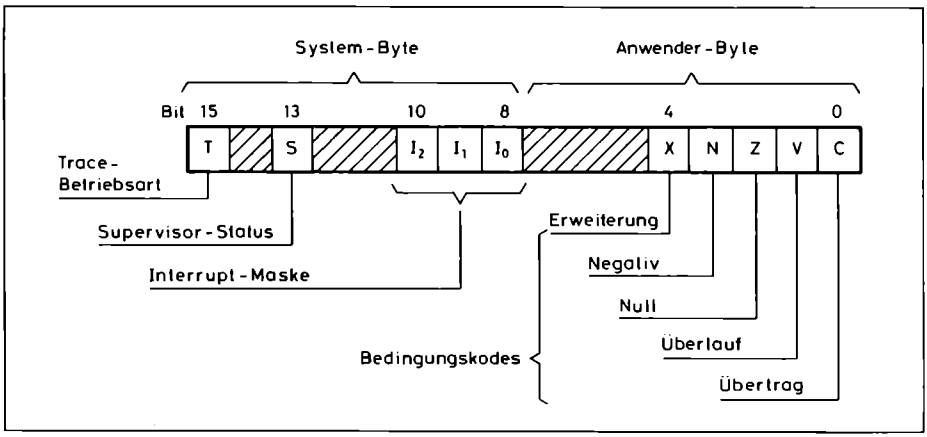
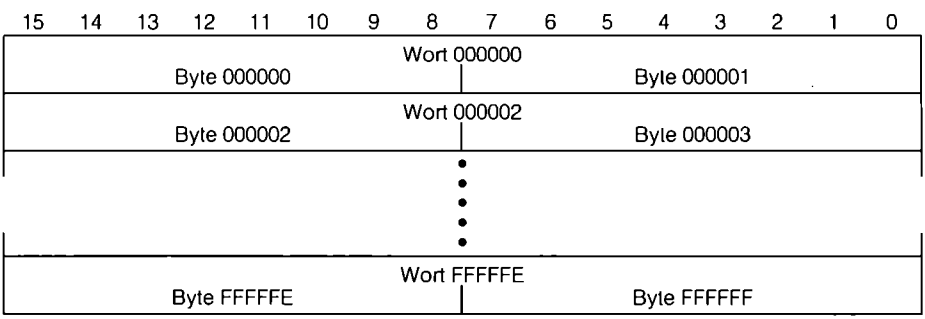


Bild 5. Statusregister des Mikroprozessors SC 68000

Bild 6. Wortorganisation im Speicher



2.3. Datenorganisation im Speicher

Die Bytes sind einzeln adressierbar, und zwar ist dem Byte in der oberen Worthälfte eine geradzahlige Adresse zugeordnet, die mit der Adresse des gesamten Wortes identisch ist (vgl. Bild 6). Die Adresse der unteren Worthälfte ist ungeradzahlig, und zwar um 1 höher als die Adresse der oberen Worthälfte oder des gesamten Wortes.

Befehle und Daten mit mehreren Bytes werden nur auf Wortbasis adressiert, d. h., sie dürfen nur geradzahlige Adressen haben. Beginnt ein Langwort mit dem 1. Wort auf der Adresse n (n geradzahlig), so hat das zugehörige 2. Wort die Adresse n + 2.

Folgende Datentypen können vom SC 68000 verarbeitet werden:

- Bits*),
- vollständige Bytes, Wörter und Langwörter (8, 16 bzw. 32 Bits),
- 32bit-Adressen,
- BCD-Ziffern.

Bild 7 zeigt, wie diese Datentypen im Speicher abgelegt werden.

2.4. Stacks und Queues

Ein Stack ist ein Speicher oder Speicherbereich, der mit Informationen (Daten) gefüllt werden kann. Beim Lesen wird die zuletzt eingegebene Information wieder entnommen, dann die davorliegende usw. (last in, first out; LIFO-Prinzip). Ein Queue wird ebenso wie der Stack mit Informationen gefüllt, jedoch wird beim Lesen die zuerst eingegebene Information entnommen, dann die danach eingegebene usw. (first in, first out; FIFO-Prinzip). In beiden Fällen wird über einen Stack pointer die jeweils anzusprechende Speicheradresse angegeben.

Der SC 68000 enthält zwei Hardware-Stack pointer, die organisatorisch den Adreßregistern zugeordnet sind. In der Betriebsart „Supervisor" kann sowohl auf das Adreßregister A7′ (SSP) als auch auf das Adreßregister A7 (USP) zugegriffen werden. In der Betriebsart „Anwender" kann nur auf das Adreßregister A7 (USP) zugegriffen werden. Ein unberechtigter Zugriff in der Betriebsart „Anwender" auf den SSP führt zu einem Trap.

Durch die universelle Verwendbarkeit der Adreßregister A0 bis A6 lassen sich außerdem softwaregesteuerte Stacks und Queues aufbauen und verwenden. Hierzu werden vor allem Befehle mit den Adressierungsarten „Adreßregister indirekt mit Postinkrement" und „Adreßregister indirekt mit Predekrement" benutzt.

*) Einzelbits sind keine Datentypen. Aus Gründen der Vereinheitlichung werden sie jedoch hier als solche behandelt.

Bild 7. Datenorganisation im Speicher

Bitdaten
1 Byte = 8 Bits

7	6	5	4	3	2	1	0

Ganzzahlige Daten
1 Byte = 8 Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB			Byte 0				LSB	Byte 1							
			Byte 2					Byte 3							

1 Wort = 16 Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB							Wort 0								LSB
							Wort 1								
							Wort 2								

1 Langwort = 32 Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB							obere Langworthälfte								
			Langwort 0												
							untere Langworthälfte								LSB
			Langwort 1												
			Langwort 2												

Adressen
1 Adresse = 32 Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB							obere Adressenhälfte								
			Adresse 0												
							untere Adressenhälfte								LSB
			Adresse 1												
			Adresse 2												

Dezimale Daten
2 BCD-Ziffern = 1 Byte

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSD	BCD 0			BCD 1			LSD	BCD 2				BCD 3			
	BCD 4			BCD 5				BCD 6				BCD 7			

MSD = höchstwertiges Bit LSD = niedrigstwertiges Bit

2.4.1. System-Stacks

Die Adreßregister A7 und A7′ stellen die sog. System Stack pointer (SP) dar, und zwar ist SP entweder der Anwender-Stack pointer (USP in A7) oder der Supervisor Stack pointer (SSP in A7′). Dies hängt im einzelnen vom Zustand des S-Bits im Statusregister ab: S = 1 (Supervisor-Zustand) bedeutet, daß der SSP in A7′ der aktive Stack pointer ist und auf den USP in A7 nicht zugegriffen wird; ein Zugriff auf den USP ist aber über die Adressierungsart „Implizit" möglich. Bei S = 0 liegen die Verhältnisse umgekehrt, jedoch kann dann nicht auf den SSP zugegriffen werden.

Jeder Stack wird von höheren zu niedrigeren Speicheradressen belegt. Mit Hilfe der Adressierungsart „Stack pointer indirekt mit Predekrement" (SP@−) wird ein weiterer Speicherplatz auf dem aktiven Stack besetzt, während mit der Adressierungsart „Stack pointer indirekt mit Postinkrement" (SP@+) ein Speicherplatz freigemacht wird.

Beim Aufruf von Subroutinen wird der Programmzähler-Inhalt in den aktiven System-Stack gerettet und von dort wieder zurückgeholt, wenn das Hauptprogramm fortgesetzt werden soll. Bei der Ausführung von Traps und Interrupts werden sowohl der Programmzähler- als auch der Statusregister-Inhalt in den Supervisor-Stack gerettet; das S-Bit ist dann automatisch gesetzt. Somit hängt die korrekte Abarbeitung von Programmteilen im Supervisor-Status nicht von Programmteilen im Anwender-Status ab.

Damit die Daten im System-Stack geordnet abgelegt werden, arbeitet dieser wortweise. Bei Verwendung von Bytes werden diese in den oberen Worthälften aufgenommen, während die jeweilige untere Worthälfte dabei unverändert bleibt.

2.4.2. Software-Stacks

Über die Adressierungsarten „Adreßregister indirekt mit Postinkrement" sowie „Adreßregister indirekt mit Predekrement" lassen sich Software-Stacks implementieren und führen. Bei Verwendung eines der Adreßregister (A0—A6) können Anwender-Stacks sowohl von hohen zu niedrigen Adressen als auch umgekehrt gefüllt werden. Die wichtigsten Gesichtspunkte sind hierbei:

- Bei Predekrementierung wird das Register dekrementiert, bevor sein Inhalt den Stack adressiert.
- Bei Postinkrementierung wird das Register inkrementiert, nachdem sein Inhalt den Stack adressiert hat.
- Daten im Byteformat müssen paarweise in den Stack eingegeben werden, wenn dies gemeinsam mit Daten im Wort- und Langwortformat erfolgen soll, damit sie geordnet wieder abgerufen werden können. Wörter und Langwörter müssen geradzahlige Adressen haben.

Soll ein Stack von hohen zu niedrigen Adressen hin betrieben werden, sind hierzu folgende Adressierungsarten erforderlich:

An@ – Schreiben von Daten in den Stack,
An@ + Lesen von Daten aus dem Stack.

Nach jedem Lesen oder Schreiben zeigt der Inhalt des Adreßregister An auf den Stack-Anfang (letzte Eintragung), wie in Bild 8 dargestellt ist.

Soll dagegen ein Stack von niedrigen zu hohen Adressen hin eingerichtet werden, ist hierzu folgendes erforderlich:

An@ + Schreiben von Daten in den Stack,
An@ – Lesen von Daten aus dem Stack.

Nach jedem Lesen oder Schreiben zeigt der Inhalt des Adreßregisters auf den nächsten verfügbaren Platz im Stack, wie aus Bild 9 zu ersehen ist.

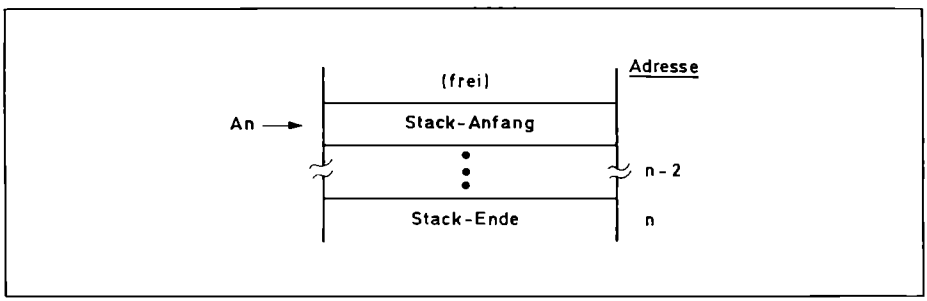
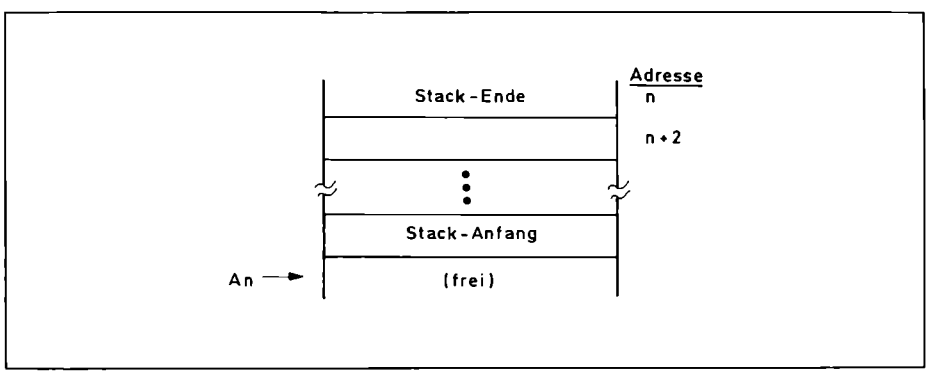


Bild 8. Betrieb eines Stacks von hohen zu niedrigen Adressen

Bild 9. Betrieb eines Stacks von niedrigen zu hohen Adressen



2.4.3. Queues

Über die Adressierungsarten „Adreßregister indirekt mit Postinkrement" sowie „Adreßregister indirekt mit Predekrement" lassen sich auch Anwender-Queues implementieren und führen, wozu der Anwender ein Adreßregister-Paar (zwei von A0—A6) benötigt. Diese zwei Pointer sind erforderlich, weil bei Queues die Daten an einem Ende eingetragen und am anderen Ende des Queues entnommen werden.

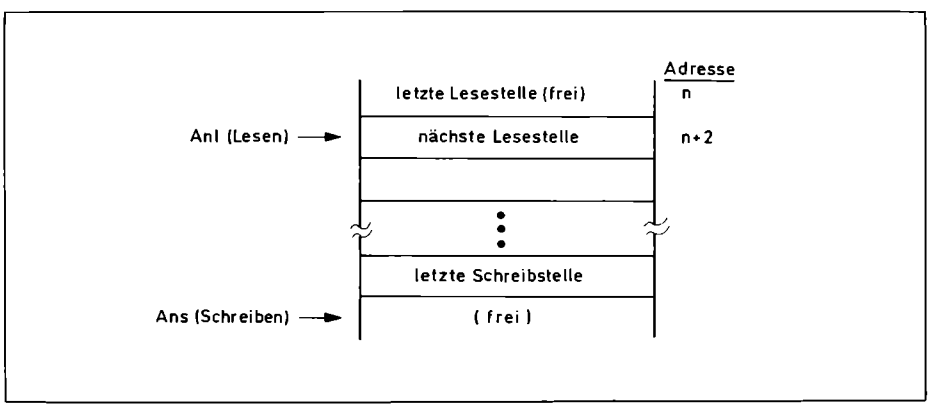
Das Queue kann sowohl von niedrigen zu hohen Adressen als auch umgekehrt betrieben werden. Im ersten Fall wird dies erreicht durch:

Ans@+ Schreiben von Daten in das Queue,
Anl@+ Lesen von Daten aus dem Queue.

Nach einer Schreiboperation zeigt das zugeordnete Adreßregister auf den nächsten verfügbaren Platz im Queue, während das den Leseoperationen zugeordnete Adreßregister unverändert auf den Platz weist, an dem die nächste Leseoperation erfolgen soll. Nach einer Leseoperation zeigt das entsprechende Adreßregister auf den nächsten Platz mit zu lesenden Daten, und das den Schreiboperationen zugeordnete Adreßregister zeigt unverändert auf den nächsten verfügbaren Platz im Queue. In Bild 10 ist dies grafisch dargestellt.

Wird das Queue als umlaufender Pufferspeicher mit endlicher Kapazität ausgelegt, so müssen die beiden Adreßpointer vor jeder Operation geprüft werden. Sobald die höchstmögliche Adresse erreicht ist, wird die Pufferlänge vom Wert des betreffenden Pointers subtrahiert.

Bild 10. Betrieb eines Queues von niedrigen zu hohen Adressen



Ein Queue-Überlauf tritt ein, wenn der Wert des Schreibpointers den Wert des Lesepointers erreicht hat. Das Queue ist leer, wenn der Wert des Lesepointers den Wert des Schreibpointers erreicht hat.

Soll das Queue in Richtung von hohen zu niedrigen Adressen betrieben werden, so ist hierzu erforderlich:

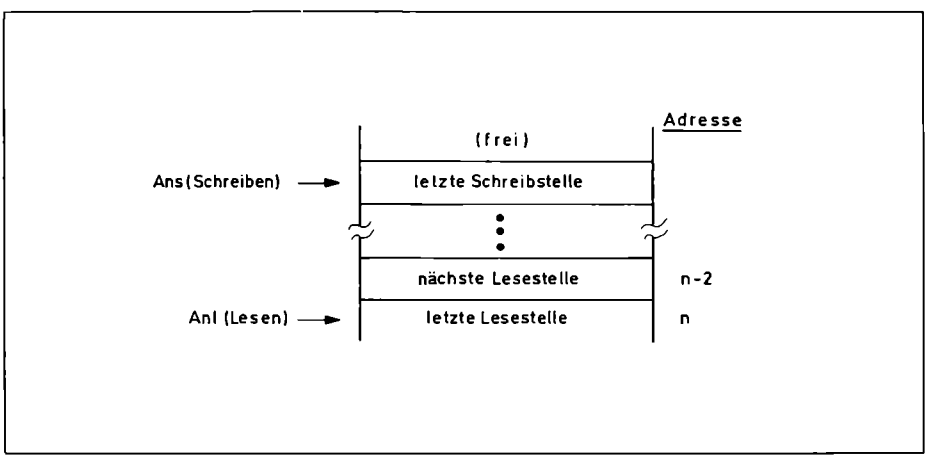
Ans@ – Schreiben von Daten in das Queue,
Anl@ – Lesen von Daten aus dem Queue.

Nach einer Schreiboperation zeigt das zugehörige Adreßregister auf den gerade belegten Platz im Queue, während das den Leseoperationen zugeordnete Adreßregister unverändert auf den Platz weist, an dem die letzte Leseoperation erfolgt ist.

Nach einer Leseoperation zeigt das entsprechende Adreßregister auf den zuletzt ausgelesenen Platz im Queue, während das den Schreiboperationen zugeordnete Adreßregister unverändert auf den Platz im Queue zeigt, der zuletzt belegt wurde. In Bild 11 ist dies grafisch dargestellt.

Wird das Queue bei dieser Betriebsweise (in Richtung von hohen zu niedrigen Adressen) als umlaufender Pufferspeicher mit endlicher Kapazität benutzt, gelten dieselben Regeln wie oben, nur muß die Pufferlänge zum betreffenden Pointer addiert werden.

Bild 11. Betrieb eines Queues von hohen zu niedrigen Adressen



3. Befehlsformate und Adressierungsarten

3.1. Bestimmung der Operandenadresse

Wie die Befehle anderer Mikroprozessoren oder -computer bestehen auch die SC 68000-Befehle aus zwei Teilen: Der erste bestimmt die Art der durchzuführenden Operation (Opcode), der zweite enthält den Operanden oder die Adresse des oder der Operanden, an dem oder an denen die Operation vorgenommen werden soll.

Die Befehle bestimmen die Operandenadresse auf drei mögliche Weisen:

- Register-Spezifikation: Die Registernummer wird im Registerfeld des Befehls angegeben; sie stellt in diesem Fall die effektive Adresse (s. Abschnitt 3.3) dar.
- Bestimmung der effektiven Adresse durch verschiedene Adressierungsarten.
- Impliziter Bezug: Die Befehle selbst bestimmen die Verwendung spezieller Register ohne weitere Angabe.

Aus dieser Einteilung resultieren 14 Adressierungsarten, die sich entsprechend Tabelle 1 in sechs Hauptgruppen gliedern:

- Register direkt,
- Register indirekt*),
- Daten absolut,
- Relativ zum Programmzähler,
- Daten unmittelbar,
- Implizit.

*) Zeigt ein Adreßregister auf eine bestimmte Speicherstelle, in der eine Adresse abgelegt ist, so wird diese Adressierungsart indirekt genannt. Das Symbol für die indirekte Adressierungsart ist die Adreßbezeichnung mit nachfolgendem @. Zum Beispiel bedeutet A4@, daß Adreßregister 4 die Speicherstelle bezeichnet, die den zu verwendenden Operanden enthält.

Bild 12. Format der SC 68000-Befehle

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Adresse
Operationswort (erstes Wort legt Operation und Modus fest)	n
Unmittelbarer Operand (falls vorhanden, ein oder zwei Wörter)	n + 2
Effektive Adreßerweiterung (Quelle) (falls vorhanden, ein oder zwei Wörter)	n + 4
Effektive Adreßerweiterung (Ziel) (falls vorhanden, ein oder zwei Wörter)	n + 6

Tabelle 1. Adressierungsarten und -kategorien

Adressierungsart	Adreßerzeugung	Effektive Adresse		Adressierungskategorien			
		Modus	Register	Daten	Speicher	Steuerung	Veränderbar
Register direkt							
Datenregister direkt	EA = Dn	000	Registernummer	×	–	–	×
Adreßregister direkt	EA = An	001	Registernummer	–	–	–	×
Register indirekt							
Adreßregister indirekt	EA = (An)	010	Registernummer	×	×	×	×
Adreßregister indirekt mit Postinkrement	EA = (An) +	011	Registernummer	×	×	–	×
Adreßregister indirekt mit Predekrement	EA = – (An)	100	Registernummer	×	×	–	×
Adreßregister indirekt mit Adreßdistanz	EA = (An) + d_{16}	101	Registernummer	×	×	×	×
Adreßregister indirekt mit Index und Adreßdistanz	EA = (An) + (Xn) + d_8	110	Registernummer	×	×	×	×
Daten absolut							
Absolut kurz	EA = (nächstes Wort)	111	000	×	×	×	×
Absolut lang	EA = (nächsten 2 Wörter)	111	001	×	×	×	×

Adressierungsart	Adreßerzeugung	Effektive Adresse		Adressierungskategorien			
		Modus	Register	Daten	Speicher	Steuerung	Veränderbar
Relativ zum Programmzähler							
Programmzähler mit Adreßdistanz	EA = (PC) + d_{16}	111	010	×	×	×	–
Programmzähler mit Index und Adreßdistanz	EA = (PC) + (Xn) + d_8	111	011	×	×	×	–
Daten unmittelbar							
Unmittelbar	EA = (PC) + 2	111	100	×	×	–	–
Unmittelbar schnell	Daten im Befehlswort						
Implizit							
Implizit Register	SR, PC, SP, USP, SSP						

Erklärung der Abkürzungen (Register-Transfer-Sprache, RTL):

EA Effektive Adresse
An Adreßregister
Dn Datenregister
Xn Adreß- oder Datenregister, verwendet als Indexregister
SR Statusregister
CCR Bedingungskoderegister (Teil des Statusregisters)

PC Programmzähler
SP Stack pointer (SSP oder USP)
USP Anwender-Stack pointer
SSP Supervisor-Stack pointer
d_8 8bit-Adreßdistanz
d_{16} 16bit-Adreßdistanz
( ) Inhalt von
N Operandengröße in Bytes (N = 1, 2, 4)

Bei den Adressierungsarten vom Typ „Register indirekt" besteht die Möglichkeit der Postinkrementierung, Predekrementierung, Adreßdistanz und Indizierung. Der Adressierungstyp „Relativ zum Programmzähler" kann ebenfalls durch Adreßdistanz und Indizierung modifiziert werden.

Wie Bild 12 zeigt, kann ein Befehl aus ein bis fünf Wörtern bestehen. Die Länge des Befehls und die durchzuführende Operation werden durch das erste Befehlswort bestimmt (sog. Operationswort). Die restlichen Wörter legen die Operanden fest und enthalten entweder unmittelbare Operanden oder zusätzliche Informationen zur effektiven Adresse (effektive Adreßerweiterung, vgl. auch Abschnitt 3.3).

3.2. Programm- und Datenbereichszuweisungen

Der Mikroprozessor SC 68000 teilt die Speicherzuweisungen in zwei Gruppen ein: Programm- und Datenbereichszuweisungen. Die Programmbereichszuweisungen beziehen sich auf denjenigen Speicherteil, der das ausführende Programm enthält, Datenbereichszuweisungen beziehen sich auf denjenigen Speicherteil, der die Daten enthält. Im allgemeinen gilt, daß alle Schreiboperationen in den Datenbereich und alle Leseoperationen aus dem Datenbereich erfolgen.

3.3. Effektive Adressen

Von den 14 in Tabelle 1 aufgelisteten Adressierungsarten erzeugen die ersten 12 eine effektive Adresse. Bei den meisten Befehlen wird die Stelle, an der der Operand abgelegt ist, durch das effektive Adreßfeld im Operationskode bestimmt. Bild 13 zeigt als Beispiel das allgemeine Format eines Befehls, der nur aus dem Operationswort besteht. Die darin enthaltene effektive Adresse ist aus zwei 3bit-Feldern aufgebaut, dem Modusfeld und dem Registerfeld. Der Inhalt des Modusfeldes bestimmt die Adressierungsart, wie in Tabelle 1 angegeben ist. Das Registerfeld legt die Nummer des zu verwendenden Registers fest.

In einigen Fällen sind zusätzliche Informationen zur effektiven Adresse erforderlich, um einen Operanden vollständig zu bestimmen. Diese als effektive Adreßerweiterung bezeichneten zusätzlichen Informationen sind im folgenden Wort oder in den folgenden Wörtern (sog. Erweiterungswort bzw. -wörter) enthalten und stellen einen Teil des Befehls dar, wie aus Bild 12 zu ersehen ist.

Bild 13. Format eines Einwort-Befehls mit effektiver Adresse

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	Effektive Adresse					
										Modus			Register		

3.3.1. Adressierungsarten Register direkt

Bei den zu dieser Gruppe gehörenden Adressierungsarten ist der Operand in einem der 16 Daten- und Adreßregister abgelegt.

Datenregister direkt

Der Operand befindet sich in dem Datenregister, das im Registerfeld der effektiven Adresse angegeben ist.

Adreßerzeugung: EA = Dn
RTL-Bezeichnung*): Dn
Modus: 000
Registernummer: n

Datenregister Dn

Operand

Adreßregister direkt

Der Operand befindet sich in dem Adreßregister, das im Registerfeld der effektiven Adresse angegeben ist.

Adreßerzeugung: EA = An
RTL-Bezeichnung: An
Modus: 001
Registernummer: n

Adreßregister An

Operand

3.3.2. Adressierungsarten Register indirekt

Bei den zu dieser Gruppe gehörenden Adressierungsarten ist der Operand im Speicher abgelegt; die Adresse des Operanden ist in dem Register enthalten, das im Registerfeld der effektiven Adresse angegeben ist.

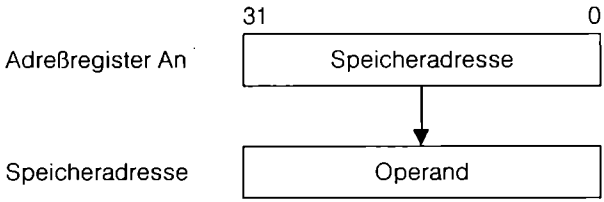
Adreßregister indirekt

Die Adresse des Operanden befindet sich in dem Adreßregister, das im Registerfeld der effektiven Adresse angegeben ist. Diese Operation gehört zur Gruppe der Daten-

*) Register Transler Language

Befehlen, mit Ausnahme der Befehle JMP (Springen) und JSR (Springen in die Unterroutine).

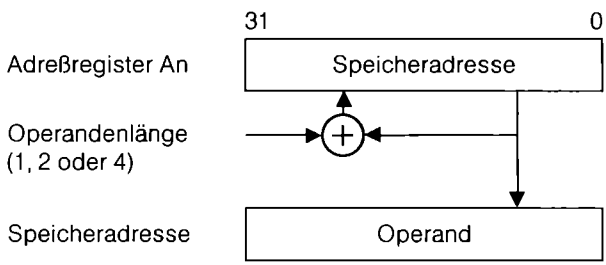
Adreßerzeugung: EA = (An)
RTL Bezeichnung: An@
Modus: 010
Registernummer: n



Adreßregister indirekt mit Postinkrement

Die Adresse des Operanden befindet sich in dem Adreßregister, das im Registerfeld der effektiven Adresse angegeben ist. Nach Durchführung der Operation wird die Operandenadresse um 1, 2 oder 4 erhöht, und zwar in Abhängigkeit von der Länge des Operanden (Byte, Wort oder Langwort). Handelt es sich bei dem Adreßregister um den Stack pointer und hat der Operand Byte-Länge, so wird die Adresse um 2 statt um 1 erhöht, damit der Stack pointer immer eine gerade Wortadresse behält. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen.

Adreßerzeugung: EA = (An)
An = An + N
RTL-Bezeichnung: An@+
Modus: 011
Registernummer: n

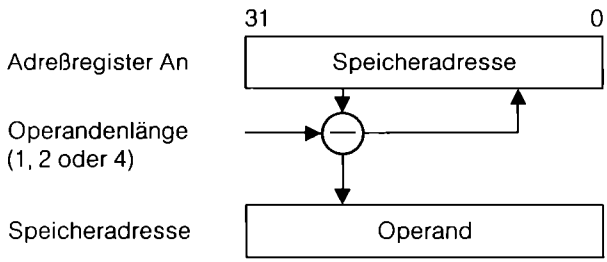


Adreßregister indirekt mit Predekrement

Die Adresse des Operanden befindet sich in dem Adreßregister, das im Registerfeld der effektiven Adresse angegeben ist. Vor Durchführung der Operation wird die Ope-

randenadresse um 1, 2 oder 4 vermindert, und zwar in Abhängigkeit von der Länge des Operanden (Byte, Wort oder Langwort). Handelt es sich bei dem Adreßregister um den Stack pointer und hat der Operand Byte-Länge, so wird die Adresse um 2 statt um 1 vermindert, damit der Stack pointer immer eine gerade Wortadresse behält. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen.

Adreßerzeugung: An = An − N
EA = (An)
RTL-Bezeichnung: An@−
Modus: 100
Registernummer: n



Adreßregister indirekt mit Adreßdistanz

Diese Adressierungsart erfordert ein Erweiterungswort. Die Adresse des Operanden ist die Summe der Adresse im Adreßregister und des vorzeichenbehafteten, ganzzahligen 16bit-Adreßdistanzwertes im Erweiterungswort. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen, mit Ausnahme der Befehle JMP (Springen) und JSR (Springen in die Subroutine).

Adreßerzeugung: EA = (An) + d_{16}
RTL-Bezeichnung: An@(d)
Modus: 101
Registernummer: n

Adreßregister indirekt mit Index und Adreßdistanz

Diese Adressierungsart erfordert ein Erweiterungswort, dessen Aufbau hierunter angegeben ist.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Register-adresse			W/L	0	0	0	Adreßdistanzwert							

Es bedeutet:

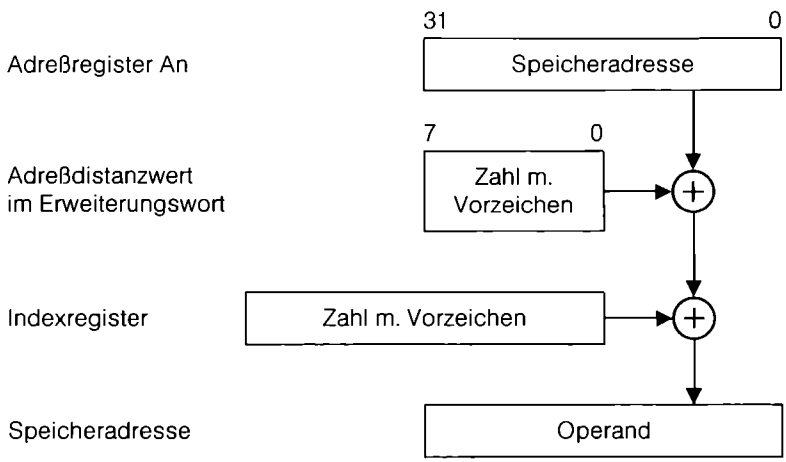
Bit 15 Indexregisterbezeichner
0 ≙ Datenregister
1 ≙ Adreßregister

Bits 14 – 12 Indexregisternummer

Bit 11 Indexgröße
0 ≙ vorzeichenbehafteter, niederwertiger 16bit-Teil im Indexregister
1 ≙ Langwort im Indexregister

Die Adresse des Operanden ist die Summe der Adresse im Adreßregister, des vorzeichenbehafteten, ganzzahligen 8bit-Adreßdistanzwertes im niederwertigen Teil des Erweiterungswortes und des Inhalts des Indexregisters. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen, mit Ausnahme der Befehle JMP (Springen) und JSR (Springen in die Subroutine).

Adreßerzeugung: $EA = (An) + (Xn) + d_8$
RTL-Bezeichnung: An@ (d,Xn.W)
An@ (d,Xn.L)
Modus: 110
Register: n



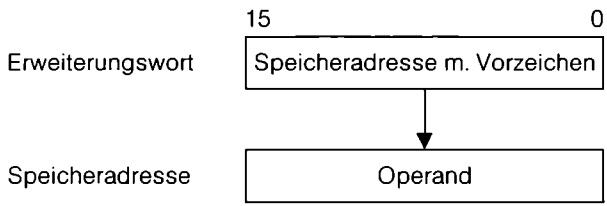
3.3.3. Adressierungsarten Daten absolut

Hier wird das Registerfeld in der effektiven Adresse zur Festlegung der speziellen Adressierungsart (anstelle der Registernummer) verwendet.

Absolute kurze Adresse

Diese Adressierungsart erfordert ein Erweiterungswort, in dem sich die vorzeichenbehaftete 16bit-Adresse des Operanden befindet. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen, mit Ausnahme der Befehle JMP (Springen) und JSR (Springen in die Subroutine).

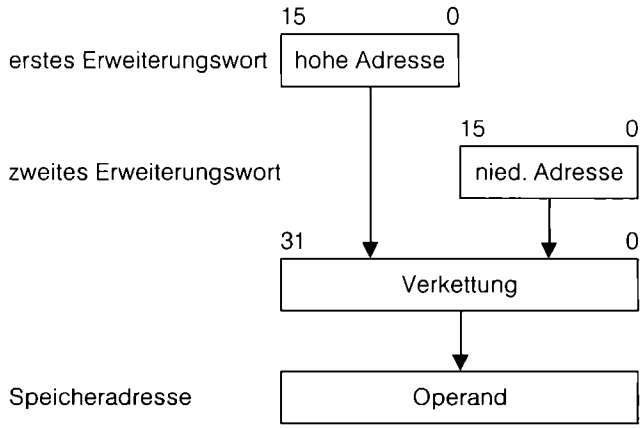
Adreßerzeugung: EA = (nächstes Wort)
RTL-Bezeichnung: xxx.W
Modus: 111
Register: 000



Absolute lange Adresse

Diese Adressierungsart erfordert zwei Erweiterungswörter. Die Adresse des Operanden wird durch Aneinanderhängen dieser beiden Erweiterungswörter gewonnen. Der höherwertige Teil der Adresse befindet sich im ersten, der niederwertige Teil im zweiten Erweiterungswort. Diese Operation gehört zur Gruppe der Datenbereichszuweisungen, mit Ausnahme der Befehle JMP (Springen) und JSR (Springen in die Subroutine).

Adreßerzeugung: EA = (nächsten 2 Wörter)
RTL-Bezeichnung: xxx.L
Modus: 111
Register: 001



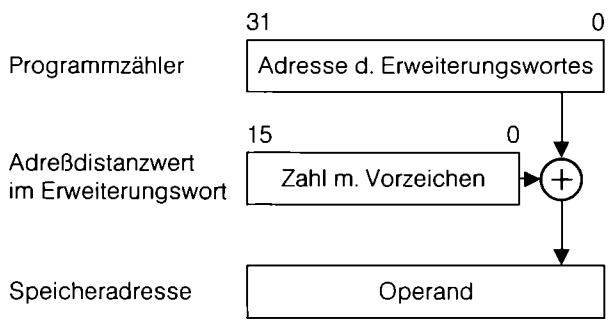
3.3.4. Adressierungsarten Relativ zum Programmzähler

Hier wird das Registerfeld in der effektiven Adresse zur Festlegung der speziellen Adressierungsart (anstelle der Registernummer) verwendet.

Programmzähler mit Adreßdistanz

Diese Adressierungsart erfordert ein Erweiterungswort. Die Adresse des Operanden ist die Summe des Programmzähler-Inhalts und des vorzeichenbehafteten, ganzzahligen 16bit-Adreßdistanzwertes im Erweiterungswort. Der Inhalt des Programmzählers ist die Adresse des Erweiterungswortes. Diese Operation gehört zur Gruppe der Programmbereichszuweisungen.

Adreßerzeugung: $EA = (PC) + d_{16}$
RTL-Bezeichnung: PC@(d)
Modus: 111
Register: 010



Programmzähler mit Index und Adreßdistanz

Diese Adressierungsart erfordert ein Erweiterungswort, dessen Aufbau hierunter angegeben ist.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Register-adresse			W/L	0	0	0	Adreßdistanzwert							

Es bedeutet:

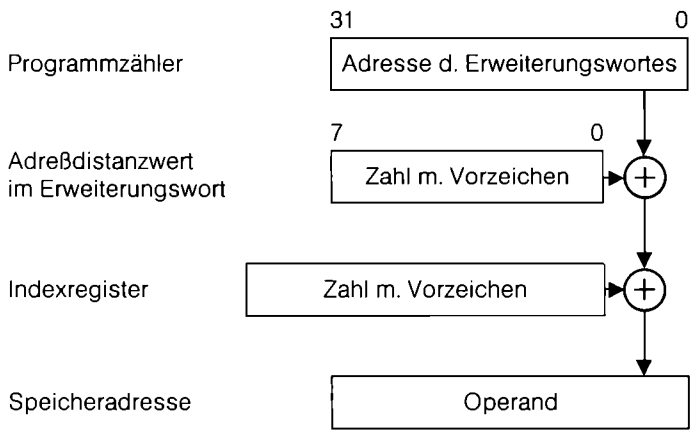
Bit 15 Indexregisterbezeichner
0 ≙ Datenregister
1 ≙ Adreßregister

Bits 14 – 12 Indexregisternummer

Bit 11 Indexgröße
0 ≙ vorzeichenbehaftet, niederwertiges Wort Zahl im Indexregister
1 ≙ Langwort, Zahl im Indexregister

Diese Adresse ist die Summe der Adresse im Programmzähler, des vorzeichenbehafteten, ganzzahligen 8bit-Adreßdistanzwertes im niederwertigen Teil des Erweiterungswortes und des Inhalts des Indexregisters. Der Inhalt des Programmzählers ist die Adresse des Erweiterungswortes. Diese Operation gehört zur Gruppe der Programmbereichszuweisungen.

Adreßerzeugung: $EA = (PC) + (Xn) + d_8$
RTL-Bezeichnung: PC@ (d,Xn.W)
PC@ (d,Xn.L)
Modus: 111
Register: 011



3.3.5. Adressierungsarten Daten unmittelbar

Unmittelbar

Diese Adressierungsart erfordert entweder ein oder zwei Erweiterungswörter je nach Länge der durchzuführenden Operationen.

Byte-Operation:
Der Operand ist das niederwertige Byte des Erweiterungswortes.

Wort-Operation:
Der Operand ist das Erweiterungswort.

Langwort-Operation:
Der Operand besteht aus zwei Erweiterungswörtern, und zwar befinden sich die höherwertigen 16 Bits im ersten und die niederwertigen 16 Bits im zweiten Erweiterungswort.

Adreßerzeugung:	EA = (PC) + 2
RTL-Bezeichnung:	#xxxx
Modus:	111
Register:	100

Format der erforderlichen Erweiterungswörter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Byte							

oder

Wort

oder

untere Hälfte
- - - Langwort - - -

3.4. Bedingungskodes und Statusregister

Die Befehle ANDI, EORI und ORI können sich mittels der effektiven Adresse auf das Bedingungskoderegister (CCR) oder das gesamte Statusregister (SR) beziehen. Eine Adressierung des gesamten Statusregisters ist nur in der Supervisor-Betriebsart möglich.

RTL-Bezeichnung:	CCR oder SR
Modus:	111
Register:	100

3.5. Impliziter Bezug auf Register

Einige Befehle beziehen sich implizit auf den Programmzähler, den System-Stack pointer (Anwender-Stack pointer oder Supervisor-Stack pointer) oder auf das Statusregister. In Tabelle 2 sind die Befehle und die zugeordneten Register aufgelistet.

4. Ausnahmeverarbeitung und -vektoren

Der Mikroprozessor SC 68000 kann sich im Zustand der normalen Befehlsverarbeitung, der Ausnahmeverarbeitung und im Haltzustand befinden (vgl. Kapitel 5 des Valvo Buches: Der 16bit-Mikroprozessor SC 68000 — Eigenschaften (Ausgabe November 1982)). Hier wird die Ausnahmeverarbeitung näher beschrieben.

4.1. Ausnahmearten

Ausnahmen können sowohl durch interne als auch durch externe Ereignisse hervorgerufen werden. Extern erzeugte Ausnahmen sind Interrupt-, Busfehler- und Rücksetzanforderungen. Interrupts werden von peripheren Einheiten initiiert und veranlassen den Prozessor, sie zu bearbeiten, während Busfehler- und Rücksetzsignale zur Steuerung des Zugriffs bzw. zum erneuten Start des Prozessors dienen. Die intern erzeugten Ausnahmen resultieren aus Befehlen, Adreßfehlern oder dem Trace-Betrieb. Bei der Ausführung der Befehle TRAP, TRAPV, CHK und DIV können Ausnahmen erzeugt werden; außerdem veranlassen unzulässige Befehle, Wortadressierung im Speicher mit einer ungeradzahligen Adresse sowie Privilegverletzungen derartige Ausnahmen. Der Trace-Modus verhält sich wie ein mit hoher Priorität intern erzeugter Interrupt nach der Ausführung eines jeden Befehls.

Tabelle 2. Befehle mit implizitem Bezug auf Register

Befehl	Mnemonic	zugeordnete Register
Verzweige bedingt Verzweige immer	B_{CC} BRA	PC
Verzweige zur Subroutine	BSR	PC, SP
Prüfe Register auf Grenzen	CHK	SSP, SR
Prüfe Bedingung, dekrementiere und verzweige	DB_{CC}	PC
Division mit Vorzeichen Division ohne Vorzeichen	DIVS DIVU	SSP, SR
Springe	JMP	PC
Springe zur Subroutine	JSR	PC, SP
Anbinden und Reservieren	LINK	SP
Transportiere zum Bedingungskoderegister Transportiere zum Statusregister	MOVE to CCR MOVE to SR	SR
Transportiere den Inhalt des Anwender-Stack pointers	MOVE USP	USP
Ablegen der effektiven Adresse	PEA	SP
Springe von Ausnahme zurück Springe zurück und lade Bedingungskodes	RTE RTR	PC, SP, SR
Springe zurück von der Subroutine	RTS	PC, SP
Trap Trap bei Überlauf	TRAP TRAPV	SSP, SR
Lösen	UNLK	SP

Bild 14. Format von Ausnahmevektoren

Wort 0	Neuer Programmzähler-Wert (oberer Adreßteil)	A0 = 0, A1 = 0
Wort 1	Neuer Programmzähler-Wert (unterer Adreßteil)	A0 = 0, A1 = 1

4.2. Ausnahmeverarbeitung

Die gesamte Ausnahmeverarbeitung erfolgt im Supervisor-Status, und zwar unabhängig vom S-Bit. Die bei der Ausnahmeverarbeitung erzeugten Buszyklen werden ebenfalls als Supervisor-Zyklen klassifiziert. Während der Ausnahmeverarbeitung wird bei allen Stack-Operationen der Supervisor-Stack pointer verwendet.

4.3. Ausnahmevektoren

Hierunter sind Speicherplätze zu verstehen, aus denen der Prozessor die Anfangsadresse eines Unterprogramms holt, das die betreffende Ausnahmeverarbeitung durchführt. Alle Ausnahmevektoren sind zwei Wörter lang (Bild 14), mit Ausnahme des Rücksetzvektors, der aus vier Wörtern besteht. Alle Ausnahmevektoren befinden sich im Supervisor-Datenbereich, ausgenommen der Rücksetzvektor, der im Supervisor-Befehlsbereich gespeichert ist. Die Adresse eines Ausnahmevektors ergibt sich aus einer 8bit-Zahl, der sogenannten Vektornummer, nach Multiplikation mit 4. Die Vektornummern werden — in Abhängigkeit vom Ausnahmeereignis — intern oder extern erzeugt. Zum Beispiel liefert eine periphere Einheit während eines Interrupt-Quittungszyklus eine 8bit-Vektornummer (Bild 15) über die Datenbusleitungen D0—D7 an den Prozessor, der sie in eine vollständige 24bit-Adresse umsetzt (s. Bild 16). Die Speicherplatzvergabe für Ausnahmevektoren ist in Tabelle 3 dargestellt.

Wie Tabelle 3 zeigt, besteht der für Ausnahmevektoren vorgesehene Speicherraum aus 512 Wörtern (1024 Bytes) und reicht von Adresse 0 bis Adresse 1023. Damit stehen 256 spezielle Vektoren zur Verfügung, von denen einige für Traps und andere Systemfunktionen reserviert sind. Von den 256 Vektoren sind 192 als Anwender-Interruptvektoren vorgesehen. Die ersten 64 Vektoren, die vorzugsweise für interne Ausnahmeverarbeitung vorgesehen sind, können je nach Ermessen des Systementwicklers auch vom Anwender als Interruptvektoren verwendet werden.

Bild 15. Format einer Vektornummer, die von einer peripheren Einheit an den Prozessor geliefert wird
v7 ist das höchstwertige Bit der Vektornummer
v0 ist das niedrigstwertige Bit der Vektornummer

D15 … D8	D7							D0
Unberücksichtigt	v7	v6	v5	v4	v3	v2	v1	v0

Bild 16. 24bit-Adresse, die aus einer 8bit-Vektornummer gewonnen wurde

A23 … A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Nullen	v7	v6	v5	v4	v3	v2	v1	v0	0	0

Tabelle 1 Ausnahmevektoren

Vektor nummer	Adresse			Zuordnung
	Dezim.	Hexadez.	Bereich	
0	0	000	SP[1])	Anfangsadresse Supervisor-Stack pointer (nach Rücksetzen)
1	4	004	SP	Anfangsadresse Programmzähler (nach Rücksetzen)
2	8	008	SD[2])	Busfehler
3	12	00C	SD	Adreßfehler
4	16	010	SD	unzulässiger Befehl
5	20	014	SD	Division durch Null
6	24	018	SD	Befehl CHK
7	28	01C	SD	Befehl TRAPV
8	32	020	SD	Privilegverletzung
9	36	024	SD	Trace
10	40	028	SD	Befehlskode 1010 Emulator
11	44	02C	SD	Befehlskode 1111 Emulator
12[3])	48	030	SD	reserviert
13[3])	52	034	SD	reserviert
14[3])	56	038	SD	reserviert
15	60	03C	SD	nicht initialisierter Interruptvektor
16—23[3])	64 95	04C 05F	SD	reserviert
24	96	060	SD	unechter Interrupt
25	100	064	SD	Interrupt-Autovektor (Ebene 1)
26	104	068	SD	Interrupt-Autovektor (Ebene 2)
27	108	06C	SD	Interrupt-Autovektor (Ebene 3)
28	112	070	SD	Interrupt-Autovektor (Ebene 4)
29	116	074	SD	Interrupt-Autovektor (Ebene 5)
30	120	078	SD	Interrupt-Autovektor (Ebene 6)
31	124	07C	SD	Interrupt-Autovektor (Ebene 7)
32—47	128 191	080 0BF	SD	Trap-Befehlsvektoren
48—63[3])	192 255	0C0 0FF	SD	reserviert
64—255	256 1023	100 3FF	SD	Anwender-Interruptvektoren

[1]) SP Programm-Stack
[2]) SD Daten-Stack
[3]) Reserviert für künftige Anwendungen. Diesen Vektornummern sollten vom Anwender keine Peripherieschaltungen zugeordnet werden.

5. Überblick über den Befehlsvorrat

Tabelle 4 enthält die 57 Grundbefehle des Mikroprozessors SC 68000 einschließlich einer symbolischen Beschreibung der betreffenden Operationen in englisch und deutsch. Außerdem gibt es eine Reihe von Befehlen, die Variationen von Grundbefehlen darstellen; sie sind in Tabelle 5 aufgelistet.

Bei der Festlegung des Befehlsvorrates wurde besonderer Wert darauf gelegt, daß die Anwendung höherer Programmiersprachen unterstützt und damit das Programmieren erleichtert wird. Von wenigen Ausnahmen abgesehen, kann jeder Befehl folgende Datentypen verarbeiten:

- Bytes (8 Bits),
- Wörter (16 Bits),
- Langwörter (32 Bits).

Viele Befehle sind für die Bearbeitung von Bits und BCD-Ziffern verwendbar.

Außerdem ist im Befehlsvorrat die Verarbeitung anderer Daten, wie z. B. von Speicheradressen oder des Statuswortes, vorgesehen.

Die meisten Befehle können jede der 14 Adressierungsarten (s. Abschnitt 3.1.) verwenden. Kombiniert man die Befehlstypen, Datentypen und Adressierungsarten, so ergeben sich mehr als 1000 sinnvolle Befehle. Dazu gehören Befehle zur Multiplikation und Division mit und ohne Vorzeichen, „schnelle" arithmetische Operationen, Arithmetik mit BCD-Ziffern sowie erweiterte Operationen durch Verwendung von Traps.

Tabelle 4. Grundbefehle des Mikroprozessors SC 68000

Mnemonic	Beschreibung	
	englisch	deutsch
ABCD	Add Decimal with Extend	Addiere dezimal mit Erweiterungsbit
ADD	Add Binary	Addiere binär
AND	AND Logical	Logisches UND
ASL	Arithmetic Shift Left	Arithmetische Verschiebung links
ASR	Arithmetic Shift Right	Arithmetische Verschiebung rechts
B_{CC}	Branch Conditionally	Verzweige bedingt
BCHG	Test a Bit and Change	Prüfe ein Bit und ändere es
BCLR	Test a Bit and Clear	Prüfe ein Bit und lösche es
BRA	Branch Always	Verzweige immer
BSET	Test a Bit and Set	Prüfe ein Bit und setze es
BSR	Branch to Subroutine	Verzweige zur Subroutine
BTST	Test a Bit	Prüfe ein Bit
CHK	Check Register against Bounds	Prüfe Register auf Grenzen
CLR	Clear an Operand	Lösche einen Operanden
CMP	Compare	Vergleiche
DB_{CC}	Test Condition, Decrement and Branch	Prüfe Bedingung, dekrementiere und verzweige
DIVS	Signed Divide	Division mit Vorzeichen
DIVU	Unsigned Divide	Division ohne Vorzeichen
EOR	Exclusive OR Logical	Logisches exklusives ODER
EXG	Exchange Registers	Vertausche Daten zwischen Registern
EXT	Sign Extend	Vorzeichenerweiterung
ILLEGAL	Illegal Instruction	Unzulässiger Befehl (TRAP)

Fortsetzung Tabelle 4. Grundbefehle des Mikroprozessors SC 68000

Mnemonic	Beschreibung	
	englisch	deutsch
JMP	Jump	Springe
JSR	Jump to Subroutine	Springe zur Subroutine
LEA	Load Effective Address	Lade die effektive Adresse
LINK	Link and Allocate	Anbinden und Reservieren
LSL	Logical Shift Left	Logische Verschiebung nach links
LSR	Logical Shift Right	Logische Verschiebung nach rechts
MOVE	Move Data from Source to Destination	Transportiere Daten von der Quelle zum Ziel
MOVEM	Move Multiple Registers	Transportiere mehrere Register
MOVEP	Move Peripheral Data	Transportiere periphere Daten
MULS	Signed Multiply	Multiplikation mit Vorzeichen
MULU	Unsigned Multiply	Multiplikation ohne Vorzeichen
NBCD	Negate Decimal with Extend	Negiere dezimal mit Erweiterungsbit
NEG	Negate	Negiere
NOP	No Operation	Keine Operation
NOT	Logical Complement	Einer-Komplement
OR	Inclusive OR Logical	Logisches ODER
PEA	Push Effective Address	Ablegen der effektiven Adresse

Fortsetzung Tabelle 4. Grundbefehle des Mikroprozessors SC 68000

Mnemonic	Beschreibung	
	englisch	deutsch
RESET	Reset External Devices	Setze externe Einheiten zurück
ROL	Rotate Left without Extend	Rundverschiebung links ohne Erweiterungsbit
ROR	Rotate Right without Extend	Rundverschiebung rechts ohne Erweiterungsbit
ROXL	Rotate Left with Extend	Rundverschiebung links mit Erweiterungsbit
ROXR	Rotate Right with Extend	Rundverschiebung rechts mit Erweiterungsbit
RTE	Return from Exception	Springe von Ausnahme zurück
RTR	Return and Restore Condition Codes	Springe zurück und lade Bedingungskodes
RTS	Return from Subroutine	Springe zurück von der Subroutine
SBCD	Subtract Decimal with Extend	Subtrahiere dezimal mit Erweiterungsbit
Scc	Set According to Condition	Setze Byte bedingt
STOP	Load Status Register and Stop	Lade das Statusregister und stoppe
SUB	Subtract Binary	Subtrahiere binär
SWAP	Swap Register Halves	Vertausche Registerhälften
TAS	Test and Set an Operand	Prüfe und setze Operanden
TRAP	Trap	Trap
TRAPV	Trap on Overflow	Trap bei Überlauf
TST	Test an Operand	Prüfe einen Operanden
UNLK	Unlink	Lösen

Tabelle 5. Variationen von Grundbefehlen des Mikroprozessors SC 68000

Mnemonic		Beschreibung	
Grund-befehl	Variation	englisch	deutsch
ADD			
	ADDA	Add Address	Addiere Adresse
	ADDI	Add Immediate	Addiere unmittelbar
	ADDQ	Add Quick	Addiere schnell
	ADDX	Add with Extend	Addiere mit Erweiterungsbit
AND			
	ANDI	AND Immediate	Logisches UND unmittelbar
CMP			
	CMPA	Compare Address	Vergleiche Adresse
	CMPI	Compare Immediate	Vergleiche unmittelbar
	CMPM	Compare Memory	Vergleiche Speicher
EOR			
	EORI	Exclusive OR Immediate	Exclusives ODER unmittelbar
MOVE			
	MOVE to CCR	Move to Condition Code Register	Transportiere zum Bedingungskoderegister
	MOVE to SR	Move to the Status Register	Transportiere zum Status-register
	MOVE from SR	Move from the Status Register	Transportiere vom Status-register
	MOVE USP	Move User Stack Pointer	Transportiere den Inhalt des Anwender-Stack pointers
	MOVEA	Move Address	Transportiere Adresse
	MOVEQ	Move Quick	Transportiere schnell
NEG			
	NEGX	Negate with Extend	Negiere mit Erweiterungsbit
OR			
	ORI	Inclusive OR Immediate	Logisches ODER unmittelbar
SUB			
	SUBA	Subtract Address	Subtrahiere Adresse
	SUBI	Subtract Immediate	Subtrahiere unmittelbar
	SUBQ	Subtract Quick	Subtrahiere schnell
	SUBX	Subtract with Extend	Subtrahiere mit Erweite-rungsbit

Die folgenden Tabellen geben einen Überblick über Form und Struktur des SC 68000-Befehlsvorrates. Die hierin enthaltenen Befehle veranlassen alle Prozessor-Funktionen, die zur Durchführung nachstehender Operationen erforderlich sind:

- Datentransport (Tabelle 6),
- arithmetische Operationen mit ganzen Zahlen (Tabelle 7),
- logische Operationen (Tabelle 8),
- Schieben und Rundschieben (Tabelle 9),
- Bitverarbeitung (Tabelle 10),
- Verarbeitung von BCD-Ziffern (Tabelle 11),
- Programmsteuerung (Tabellen 12 und 13),
- Systemsteuerung (Tabelle 14).

Tabelle 6. Datentransportoperationen

Befehl	Format des Operanden	Operation
EXG	32	Rx ↔ Ry
LEA	32	EA → An
LINK	–	An → SP@ – SP → An SP + d → SP
MOVE	8, 16, 32	(EA)s → EAd
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA
MOVEP	16, 32	(EA) → Dn Dn → EA
MOVEQ	8	# xxx → Dn
PEA	32	EA → SP@ –
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → SP SP@ + → An

Erklärung der Abkürzungen

s	Quelle	@ –	indirekt mit Predekrement
d	Ziel	@ +	indirekt mit Postinkrement
[ ]	Bitnummern	#	unmittelbar

Tabelle 7. Arithmetische Operationen mit ganzen Zahlen

Befehl	Format des Operanden	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) → Dn (EA) + Dn → EA (EA) + #xxx → EA An + (EA) → An
ADDX	8, 16, 32 16, 32	Dx + Dy + X → Dx Ax@− + Ay@− + X → Ax@
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32 16, 32	Dn − (EA) (EA) − #xxx Ax@ + − Ay@ + An − (EA)
DIVS	32 + 16	Dn/(EA) → Dn
DIVU	32 + 16	Dn/(EA) → Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 × 16 → 32	Dn × (EA) → Dn
MULU	16 × 16 → 32	DN × (EA) → Dn
NEG	8, 16, 32	0 − (EA) → EA
NEGX	8, 16, 32	0 − (EA) − X − EA
SUB	8, 16, 32 16, 32	Dn − (EA) → Dn (EA) − Dn → EA (EA) − #xxx → EA An − (EA) → An
SUBX	8, 16, 32	Dx − Dy − X → Dx Ax@− − Ay@− − X → Ax@
TAS	8	(EA) − 0, 1 → EA[7]
TST	8, 16, 32	(EA) − 0

Tabelle 8. Logische Operationen

Befehl	Format des Operanden	Operation
AND	8, 16, 32	Dn ∧ (EA) → Dn (EA) ∧ Dn → EA (EA) ∧ # xxx → EA
OR	8, 16, 32	Dn v (EA) → Dn (EA) v Dn → EA (EA) v # xxx → EA
EOR	8, 16, 32	(EA) ≡ Dy → EA (EA) ≡ # xxx → EA
NOT	8, 16, 32	~ (EA) → EA

Tabelle 9. Schiebe- und Rundschiebeoperationen

Befehl	Format des Operanden	Operation
ASL	8, 16, 32	X/C ← ← 0
ASR	8, 16, 32	→ → X/C
LSL	8, 16, 32	X/C ← ← 0
LSR	8, 16, 32	0 → → X/C
ROL	8, 16, 32	C ← ←
ROR	8, 16, 32	→ → C
ROXL	8, 16, 32	C ← ← X
ROXR	8, 16, 32	X → → C

Tabelle 10. Bitmanipulations-Operationen

Befehl	Format des Operanden	Operation
BTST	8, 32	~ Bit von (EA) → Z
BSET	8, 32	~ Bit von (EA) → Z 1 → Bit von EA
BCLR	8, 32	~ Bit von (EA) → Z 0 → Bit von EA
BCHG	8, 32	~ Bit von (EA) → Z ~ Bit von (EA) → Bit von EA

Tabelle 11. Operationen mit BCD-Ziffern

Befehl	Format des Operanden	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $Ax@ -_{10} + Ay@ -_{10} + X \rightarrow Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $Ax@ -_{10} - Ay@ -_{10} - X \rightarrow Ax@$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

Tabelle 12. Programmsteuerungs-Operationen

Befehl	Operation
Bedingte Sprünge	
B_{CC}	Verzweige bedingt (14 Bedingungen) Adreßdistanzwerte: 8 und 16 bit
DB_{CC}	Prüfe Bedingung, dekrementiere und verzweige Adreßdistanzwert: 16 bit
S_{CC}	Setze Byte bedingt (16 Bedingungen)
Unbedingte Sprünge	
BRA	Verzweige immer Adreßdistanzwerte: 8 und 16 bit
BSR	Verzweige zur Subroutine Adreßdistanzwerte: 8 und 16 bit
JMP	Springe
JSR	Springe zur Subroutine
Rücksprünge	
RTR	Springe zurück und lade Bedingungskodes
RTS	Springe zurück von der Subroutine

Tabelle 13. Mögliche Bedingungen bei bedingten Sprungbefehlen

Bedingung	Kode	Beschreibung		Prüfung
		englisch	deutsch	
CC	0100	carry clear	Übertragsbit zurückgesetzt	C = 0?
CS	0101	carry set	Übertragsbit gesetzt	C = 1?
EQ	0111	equal	gleich	Z = 1?
F[1])	0001	false	falsch	niemals wahr
GE[2])	1100	greater or equal	größer oder gleich	N ≢ V = 0?
GT[2])	1110	greater	größer	Zv (N ≢ V) = 0?
HI	0010	higher	höher	CvZ = 0?
LE[2])	1111	less or equal	kleiner oder gleich	Zv (N ≢ V) = 1?
LS	0011	lower or same	niedriger oder identisch	CvZ = 1?
LT[2])	1101	less	weniger	N ≢ V = 1?
MI	1011	minus	negativ	N = 1?
NE	0110	not equal	ungleich	Z = 0?
PL	1010	plus	positiv	N = 0?
T[1])	0000	true	wahr	immer wahr
VC	1000	no overflow	kein Überlauf	V = 0?
VS	1001	overflow	Überlauf	V = 1?

[1]) Nicht anwendbar für den Befehl B_{CC}
[2]) Zweier-Komplement bei Sprungbedingungen

v ODER
≢ Exklusives ODER (Antivalenz)

Tabelle 14. Systemsteuerungs-Operationen

Befehl	Operation
Privilegiert	
RESET	Setze externe Einheiten zurück
RTE	Springe von Ausnahme zurück
STOP	Lade Statusregister und stoppe
ORI to SR	Logisches ODER zum Statusregister
MOVE USP	Transportiere den Inhalt des Anwender-Stack pointers
ANDI to SR	Logisches UND zum Statusregister
EORI to SR	Logisches Exclusiv ODER zum Statusregister
MOVE EA to SR	Lade effektive Adresse ins Statusregister
Trap-Erzeugung	
TRAP	Trap
TRAPV	Trap bei Überlauf
CHK	Prüfe Register auf Grenzen
Statusregister	
ANDI to CCR	Logisches UND zum Bedingungskode
EORI to CCR	Logisches Exclusiv ODER zum Bedingungskode
MOVE EA to CCR	Lade Bedingungskode
ORI to CCR	Logisches ODER zum Bedingungskode
MOVE SR to EA	Lies Statusregister

6. Beschreibung des Befehlsvorrats

6.1. Erläuterungen zur Befehlsbeschreibung

Im folgenden werden die bei der Befehlsbeschreibung verwendeten Unterabschnitte an einem Beispiel erläutert:

Zu beschreibende Größe	Erläuterungen
ABCD	Mnemonic
Add Decimal with Extend Addiere dezimal mit Erweiterungsbit	Erklärung in Englisch und Deutsch
Operation: $(\text{Ziel})_{10} + (\text{Quelle})_{10} + X \rightarrow \text{Ziel}$	Befehlsbeschreibung in RTL*) (hier Dezimalarithmetik)
ABCD Dy nach Dx ABCD Ay@− nach Ax@−	verschiedene Verwendungsmöglichkeiten für diesen Befehl
Assembler Syntax: ABCD Dy, Dx ABCD −(Ay), −(Ax)	Assembler-Syntax entsprechend den Verwendungsmöglichkeiten dieses Befehls
Operandenlänge: Byte	Ist nur eine Operandenlänge angegeben, so ist diese implizit durch den Befehl vorgegeben, anderenfalls muß sie explizit im Befehl enthalten sein.
Beschreibung: Der Quellenoperand und . . .	verbale Befehlsbeschreibung
Bedingungskodes: X N Z V C ★ U ★ U ★ N nicht definiert Z wird gelöscht, wenn . . .	Angabe, in welcher Weise die einzelnen Bits des Bedingungskodes beeinflußt werden.

*) Register Transfer Language

Befehlsformat

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Register Rx	1	0	0	0	0	R/M	Register Ry

Bitmuster und Felder des Befehlswortes sowie alle weiteren Wörter, die Teil des Befehls sind. Erweiterungen der effektiven Adresse werden nicht explizit angegeben; falls vorhanden, ergeben sie sich aus dem dargestellten Befehl. Bei Transportbefehlen steht die Erweiterung der effektiven Adresse der Quelle an erster, die des Ziels an zweiter Stelle.

Befehlsfelder

Registerfeld Rx bezeichnet das . . . Bedeutung und zugelassene Werte bestimmter Felder des Befehlswortes

6.2. Abkürzungen und Definitionen bei der Befehlsbeschreibung

Folgende Abkürzungen und Definitionen werden bei der detaillierten Befehlsbeschreibung verwendet:

Operanden

An	Adreßregister	CCR	Bedingungskoderegister (unteres Byte des Statusregisters)
Dn	Datenregister	SSP	Supervisor-Stack pointer (≙ A 7′)
Rn	beliebiges Daten- oder Adreßregister	USP	Anwender-Stack pointer (≙ A 7)
PC	Programmzähler	SP	aktiver Stack pointer (SSP oder USP)
SR	Statusregister	s	Operandengröße

Unmittelbare Daten	unmittelbare Daten aus dem Befehl
d	Adreßdistanz
Quelle	Speicherplatz oder Register, in dem sich der zu verarbeitende Operand befindet
Ziel	Speicherplatz oder Register, in dem das Ergebnis der Operanden-Verarbeitung abgelegt ist
Vektor	Speicherplatz, der Anfangsadresse einer Subroutine für Ausnahmeverarbeitung enthält

Bedingungskodes

X	Erweiterung	★	Bit wird beeinflußt
N	Negativ	–	Bit wird nicht beeinflußt
Z	Null	0	Bit wird 0
V	Überlauf	1	Bit wird 1
C	Übertrag	U	nicht definiert

Unterfelder und Zeichenerläuterung

<bit> aus einem <operand>	bezeichnet ein einzelnes Bit des Operanden
<operand> [<Bitnummer> : <Bitnummer>]	bezeichnet ein Unterfeld eines Operanden
(<operand>)	Inhalt des betreffenden Speicherplatzes oder Registers
<operand>$_{10}$	Operand ist eine binär kodierte Dezimalzahl; Operationen werden im Dezimalsystem durchgeführt
<operand> @ <mode>	Indirekter Registeroperator, der anzeigt, daß das Operandenregister auf den Speicherplatz des Befehlsoperanden zeigt

Operationen

Alle Operationen lassen sich einteilen in Operationen, bei denen

- zwei Operanden beteiligt sind,
- ein Operand beteiligt ist,
- sonstige Operationen durchgeführt werden.

Operationen mit zwei Operanden

Diese Operationen werden geschrieben <operand> <op> <operand>, wobei <op> eine der folgenden Operationen bedeutet:

→	Der linke Operand wird zu demjenigen Platz transportiert, der durch den rechten Operanden angegeben ist.
↔	Die Inhalte der beiden Operanden werden miteinander vertauscht.
+	Die Operanden werden addiert.
−	Der rechte Operand wird vom linken subtrahiert.
×	Die Operanden werden multipliziert.
/	Der erste Operand wird durch den zweiten dividiert.
∧	Die Operanden werden durch ein logisches UND verknüpft.
∨	Die Operanden werden durch ein logisches ODER verknüpft.
≢	Die Operanden werden durch exklusives ODER verknüpft.
<	Vergleichstest; wahr, wenn der linke Operand kleiner als der rechte ist.
>	Vergleichstest; wahr, wenn der linke Operand größer als der rechte ist.
≠	Vergleichstest; wahr, wenn die beiden Operanden verschieden sind.
verschoben um rundgeschoben um	Der linke Operand wird um die im rechten Operanden angegebene Stellenzahl verschoben bzw. rundgeschoben.

Operationen mit einem Operanden

~ <operand>	Der Operand wird logisch komplementiert.
<operand> sign extended	Der Operand wird vorzeichenerweitert; alle Bits der oberen Hälfte werden dem höchstwertigen Bit (Vorzeichen) der unteren Hälfte gleichgesetzt.
<operand> tested	Der Operand wird mit Null verglichen; aufgrund der Ergebnisse werden die Bedingungskodes gesetzt.

Sonstige Operationen

TRAP	ist gleichbedeutend mit PC → SSP@ − ; SR → SSP@; (Vektor) → PC.
STOP	Der Prozessor wird in den gestoppten Zustand übergeführt und wartet auf Interrupts.
wenn <Bedingung> dann <Operationen> sonst <Operationen>	Die Bedingung wird geprüft. Ist sie wahr, werden die Operationen hinter „dann" ausgeführt. Ist sie unwahr und ein Befehlsteil „sonst" vorhanden, so werden die darin enthaltenen Operationen ausgeführt. Ist die Bedingung unwahr und ein Befehlsteil „sonst" nicht vorhanden, so führt der Befehl zu keiner Operation.

6.3. Befehlsbeschreibung in alphabetischer Reihenfolge

Die folgende Befehlsbeschreibung enthält sowohl die 57 Grundbefehle als auch die Variationen der Grundbefehle, wie sie in Tabelle 5 (Abschnitt 5) aufgelistet sind.

ABCD

Add Decimal with Extend

Addiere dezimal mit Erweiterungsbit

Operation: $(\text{Ziel})_{10} + (\text{Quelle})_{10} + X \rightarrow \text{Ziel}$
ABCD Dy zu Dx
ABCD Ay@− zu Ax@−

Assembler Syntax: ABCD Dy, Dx
ABCD −(Ay), −(Ax)

Operandenlänge: Byte

Beschreibung:

Der Quellenoperand und das Erweiterungsbit X werden zum Zieloperanden addiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Addition wird mit binär kodierten Dezimalziffern durchgeführt. Für die Adressierung der Operanden gibt es zwei Möglichkeiten:

1. Von Datenregister zu Datenregister. Die im Befehl angegebenen Datenregister enthalten die Operanden.
2. Von Speicher zu Speicher. Die Operanden werden in der Adressierungsart „Adreßregister indirekt mit Predekrement" unter Verwendung der im Befehl angegebenen Adreßregister adressiert.

Der Befehl wird nur in der Operandenlänge „Byte" ausgeführt.

Bedingungskodes:

X	N	Z	V	C
★	U	★	U	★

N nicht definiert.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V nicht definiert.
C wird gesetzt, wenn ein dezimaler Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	0	0	Register Rx	1	0	0	0	0	R/M	Register Ry

Befehlsfelder:

Registerfeld Rx	bezeichnet das Zielregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")
Bit R/M	bezeichnet die Adressierungsart des Operanden: 0 ≙ Operation erfolgt von Datenregister zu Datenregister 1 ≙ Operation erfolgt von Speicher zu Speicher
Registerfeld Ry	bezeichnet das Quellenregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")

ADD

Add Binary

Addiere binär

Operation:

(Ziel) + (Quelle) → Ziel
ADD <ea> zu Dn
ADD Dn zu <ea>

Assembler Syntax:

ADD.s <ea>, Dn
ADD.s Dn, <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird zum Zieloperanden addiert und der Zieloperand mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Dort wird auch angegeben, wo Quelle und Ziel der Operanden sind.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	1	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld — bestimmt eines der acht Datenregister.

Operationsmodusfeld

Byte	Wort	Langwort	Operation
000	001	010	(<Dn>) + (<ea>) → <Dn>
100	101	110	(<ea>) + (<Dn>) → <ea>

Effektives Adreßfeld — bestimmt die Adressierungsart.

a) Wenn dieses Feld den Quellenoperanden adressiert, sind alle Adressierungskategorien erlaubt. Hat der Operand allerdings nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

b) Wenn dieses Feld den Zieloperanden adressiert, ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

ADDA

Add Address

Addiere Adresse

Operation: (Ziel) + (Quelle) → Ziel
ADD <ea> zu An

Assembler Syntax: ADDA.s <ea> An

Operandenlänge: s = (Wort, Langwort)

Beschreibung:

Der Quellenoperand wird zum Zieladreßregister addiert und das Ergebnis im Adreßregister abgelegt. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	1	Register	Operations-modus	Effektive Adresse

Add Address

Addiere Adresse

ADDA

Befehlsfelder:

Registerfeld	bestimmt eines der acht Adreßregister. Dies ist immer das Zielregister.
Operationsmodusfeld	bestimmt die Länge der Operation: 011 ≙ Wortoperation. Der Quellenoperand wird durch Vorzeichenerweiterung zu einem Langwort. Die Operation erfolgt im Adreßregister unter Verwendung aller 32 Bits. 111 ≙ Langwortoperation
Effektives Adreßfeld	bestimmt den Quellenoperanden. Alle Adressierungskategorien sind zulässig.

ADDI

Add Immediate

Addiere unmittelbar

Operation: (Ziel) + unmittelbare Daten → Ziel
ADD # <data> zu <ea>

Assembler Syntax: ADDI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden zum Zieloperanden addiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" (s. Tabelle 1) zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamten unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

ADDQ

Add Quick

Addiere schnell

Operation: (Ziel) + unmittelbare Daten → Ziel
ADDQ # <data> zu <ea>

Assembler Syntax: ADDQ.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden zum Zieloperanden addiert. Der Datenbereich umfaßt 1 bis 8 (Bits 9 bis 11). Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Wort- und Langwortoperationen dürfen auch mit den Adreßregistern durchgeführt werden.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Die Bedingungskodes werden nicht beeinflußt, wenn zum Inhalt eines Adreßregisters addiert wird.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Daten			0	Länge		Effektive Adresse					

Add Quick

Addiere schnell

ADDQ

Befehlsfelder:

Datenfeld	Drei Bits (9 bis 11) mit direkten Daten, entsprechend einem Datenbereich von 1 bis 8 (8 ≙ 000).
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Veränderbar" zulässig. Hat der Operand nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

ADDX

Add with Extend

Addiere mit Erweiterungsbit

Operation:

(Ziel) + (Quelle) + X → Ziel
ADDX Dy zu Dx
ADDX Ay@− zu Ax@−

Assembler Syntax:

ADDX.s Dy, Dx
ADDX.s −(Ay), −(Ax)

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand und das Erweiterungsbit werden zum Zieloperanden addiert; der Zieloperand wird mit dem Ergebnis überschrieben. Für die Adressierung der Operanden gibt es zwei Möglichkeiten:

1. Von Datenregister zu Datenregister. Die im Befehl angegebenen Datenregister enthalten die Operanden.
2. Von Speicher zu Speicher. Die Operanden werden in der Adressierungsart „Adreßregister indirekt mit Predekrement" unter Verwendung der im Befehl angegebenen Adreßregister adressiert.

Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	0	1	Register Rx	1	Länge	0	0	R/M	Register Ry

Befehlsfelder:

Registerfeld Rx	bezeichnet das Zielregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit R/M	bezeichnet die Adressierungsart des Operanden: 0 ≙ Operation erfolgt von Datenregister zu Datenregister 1 ≙ Operation erfolgt von Speicher zu Speicher
Registerfeld Ry	bezeichnet das Quellenregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")

AND

AND Logical

Logisches UND

Operation:

(Ziel) ∧ (Quelle) → Ziel
AND <ea> zu Dn
AND Dn zu <ea>

Assembler Syntax:

AND.s <ea>, Dn
AND.s Dn, <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird mit dem Zieloperanden über ein logisches UND verknüpft und der Zieloperand mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt eines Adreßregisters darf nicht als Operand verwendet werden.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gelöscht.
C wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	1	0	0	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld	bestimmt eines der acht Datenregister.
Operationsmodusfeld	
Effektives Adreßfeld	bestimmt die Adressierungsart. a) Wenn dieses Feld den Quellenoperanden adressiert, ist nur die Adressierungskategorie „Daten" zulässig. b) Wenn dieses Feld den Zieloperanden adressiert, ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

Byte	Wort	Langwort	Operation
000	001	010	(<Dn>) ∧ (<ea>) → <Dn>
100	101	110	(<ea>) ∧ (<Dn>) → <ea>

ANDI

AND Immediate

Logisches UND unmittelbar

Operation: (Ziel) ∧ unmittelbare Daten → Ziel
AND # <data> zu <ea>

Assembler Syntax: ANDI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden mit dem Zieloperanden über ein logisches UND verknüpft; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher. Zugriff zum Statusregister ist auf zwei Arten möglich:

1. Wenn die Operandenlänge ein Byte beträgt, beeinflußt die Operation nur das untere Byte des Statusregisters, das die Bedingungskodes enthält.
2. Wenn die Operandenlänge ein Wort beträgt, erstreckt sich die Operation auf das gesamte Statusregister. Dies ist eine privilegierte Operation.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C wird gelöscht.
X nicht beeinflußt.

Wenn das Statusregister adressiert wird, werden die Bedingungskodes entsprechend der logischen Verknüpfung beeinflußt.

AND Immediate

Logisches UND unmittelbar

ANDI

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es sind nur die Adressierungskategorie „Daten ∧ Veränderbar" sowie die Adressierungsart „Unmittelbar" (ausschließlich bezogen auf das Statusregister) zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamtem unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

ASL, ASR

Arithmetic Shift

Arithmetische Verschiebung

Operation:

(Ziel) verschoben um <Stellenzahl> → Ziel
ASd Dy durch Dx
ASd Dy durch # <data>
ASd <ea>

Assembler Syntax:

ASd.s Dx, Dy
ASd.s # <data>, Dy
ASd <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

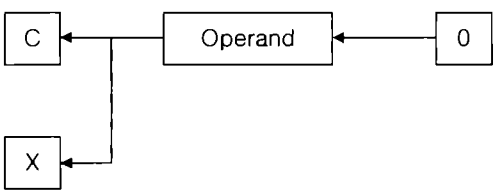
Beschreibung:

Die Bits des Operanden werden in der angegebenen Richtung arithmetisch verschoben. Das zuletzt aus dem Operanden geschobene Bit wird zum Übertragsbit. Die Anzahl der verschobenen Stellen in einem Register kann auf zwei Arten angegeben werden:

1. Unmittelbar: Die Stellenzahl ist im Befehl angegeben (1 bis 8 Stellen, wobei 8 ≙ 000).
2. Register: Die Stellenzahl ist in einem Datenregister enthalten, das im Befehl angegeben ist.

Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt einer Speicherstelle kann nur um ein Bit verschoben werden, wobei die Operandenlänge auf ein Wort beschränkt ist.

ASL: Der Operand wird nach links um die angegebene Stellenzahl verschoben. Das aus der höchstwertigen Stelle geschobene Bit wird sowohl zum Übertrags- als auch zum Erweiterungsbit; in die niedrigstwertige Stelle wird eine Null nachgeschoben. Das Überlaufbit zeigt jeden Vorzeichenwechsel während der Verschiebung an.

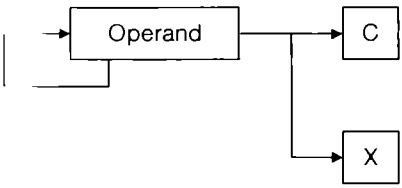


Arithmetic Shift

Arithmetische Verschiebung

ASL, ASR

ASR: Der Operand wird nach rechts um die angegebene Stellenzahl verschoben. Das aus der niedrigstwertigen Stelle geschobene Bit wird sowohl zum Übertrags- als auch zum Erweiterungsbit; das Vorzeichenbit wird in die höchstwertige Stelle reproduziert.



Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.

Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.

V wird gesetzt, wenn das höchstwertige Bit zu irgendeiner Zeit während der Verschiebung geändert wird. Anderenfalls wird V gelöscht.

C wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, wird C gelöscht.

X wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, bleibt X unbeeinflußt.

Befehlsformat (Verschiebung eines Registers):

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Stellenzahl/ Register	dr	Länge	i/r	0	0	Register

ASL, ASR

Arithmetic Shift

Arithmetische Verschiebung

Befehlsfelder:

Stellenzahl-/Registerfeld	bestimmt die Stellenzahl oder das Register, in dem sich die Stellenzahl befindet. i/r = 0: Die Bits 9 bis 11 enthalten die Stellenzahl; die Werte 0 bis 7 entsprechen einem Bereich von 8, wobei 8 ≙ 000. i/r = 1: Die Bits 9 bis 11 geben das Datenregister an, in dem sich die Stellenzahl (modulo 64) befindet.
Bit dr	gibt die Verschieberichtung an: 0 ≙ Verschiebung nach rechts 1 ≙ Verschiebung nach links
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit i/r	i/r = 0: Stellenzahl ist im Befehl enthalten (unmittelbare Daten). i/r = 1: Stellenzahl befindet sich in dem Register, das die Bits 9 bis 11 bezeichnet.
Registerfeld	gibt das Datenregister an, dessen Inhalt verschoben werden soll.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	Effektive Adresse					

Befehlsfelder (Verschiebung eines Speichers):

Bit dr	gibt die Verschieberichtung an: 0 ≙ Verschiebung nach rechts 1 ≙ Verschiebung nach links
Effektives Adreßfeld	bestimmt den Operanden, der verschoben werden soll. Es ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

Bcc

Branch Conditionally

Verzweige bedingt

Operation: Wenn $_{CC}$, dann PC + d → PC
B_{CC} <label>

Assembler Syntax: B_{CC} <label>

Operandenlänge: Byte, Wort

Beschreibung:

Ist die angegebene Bedingung erfüllt, so wird die Programmausführung an der Stelle PC + Adreßdistanz fortgesetzt. Die Adreßdistanz ist ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes angibt. Der Programmzähler enthält die gegenwärtige Befehlsadresse plus 2. Wenn die 8bit-Adreßdistanz im Befehlswort Null ist, wird die 16bit-Adreßdistanz verwendet, die unmittelbar auf das Befehlswort folgt. Die Bedingungen „$_{CC}$" sind folgendermaßen definiert:

CC	0100	Übertragsbit zurückgesetzt	LS	0011	niedriger oder identisch
CS	0101	Übertragsbit gesetzt	LT	1101	weniger
EQ	0111	gleich	MI	1011	negativ
GE	1100	größer oder gleich	NE	0110	ungleich
GT	1110	größer	PL	1010	positiv
HI	0010	höher	VC	1000	kein Überlauf
LE	1111	kleiner oder gleich	VS	1001	Überlauf

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Branch Conditionally

Verzweige bedingt

Bcc

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Bedingung				8bit-Adreßdistanzwert							
16bit-Adreßdistanzwert, falls 8bit-Adreßdistanzwert Null ist															

Befehlsfelder:

Bedingungsfeld	enthält eine der 14 Bedingungen.
8bit-Adreßdistanzfeld	enthält ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes zwischen dem Verzweigungsbefehl und dem danach auszuführenden Befehl (bei Erfüllung der Bedingung) angibt.
16bit-Adreßdistanzfeld	gestattet einen größeren Adreßdistanzwert als 8 bit. Es wird nur verwendet, wenn der 8bit-Adreßdistanzwert Null ist.

BCHG

Test a Bit and Change

Prüfe ein Bit und ändere es

Operation:

~(<Bitnummer> vom Ziel) → Z
~(<Bitnummer> vom Ziel) → <Bitnummer> vom Ziel
BCHG Dn von <ea>
BCHG # <data> von <ea>

Assembler Syntax:

BCHG Dn, <ea>
BCHG # <data>, <ea>

Operandenlänge: Byte, Langwort

Beschreibung:

Ein einzelnes Bit des Zieloperanden wird getestet; sein Wert bestimmt das Bit Z im Bedingungskode. Nach dem Test wird dieses Bit im Zieloperanden geändert. Ist der Zieloperand in einem Datenregister, so erfolgt die Bitnumerierung modulo 32, wodurch die Manipulation aller Bits in einem Datenregister möglich wird. Ist der Zieloperand in einer Speicherstelle, so wird ein Byte aus dieser Stelle gelesen, die Bitoperation unter Verwendung der Bitnummer modulo 8 durchgeführt und das Byte wieder an die Speicherstelle zurückgeschrieben. Die Bitnummer kann auf zwei Arten festgelegt sein:

1. Unmittelbar: Die Bitnummer ist in einem zweiten Wort des Befehls angegeben.
2. Register: Die Bitnummer ist in einem Datenregister enthalten, das im Befehl bezeichnet wird.

Bedingungskodes:

X	N	Z	V	C
–	–	★	–	–

N nicht beeinflußt.
Z wird gesetzt, wenn das getestete Bit Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C, X nicht beeinflußt.

Befehlsformat (Bitnummer dynamisch):

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	0	0	0	Register	1	0	1	Effektive Adresse

Test a Bit and Change

Prüfe ein Bit und ändere es

BCHG

Befehlsfelder:

Registerfeld	bezeichnet das Datenregister, das die Bitnummer enthält.
Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Befehlsformat (Bitnummer statisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	Effektive Adresse					
0	0	0	0	0	0	0	0	Bitnummer							

Befehlsfelder:

Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Bitnummerfeld	bezeichnet die Bitnummer.

BCLR

Test a Bit and Clear

Prüfe ein Bit und lösche es

Operation:

~ (<Bitnummer> vom Ziel) → Z
0 → <Bitnummer> vom Ziel
BCLR Dn von <ea>
BCLR # <data> von <ea>

Assembler Syntax:

BCLR Dn, <ea>
BCLR # <data>, <ea>

Operandenlänge: Byte, Langwort

Beschreibung:

Ein einzelnes Bit des Zieloperanden wird getestet; sein Wert bestimmt das Bit Z im Bedingungskode. Nach dem Test wird dieses Bit im Zieloperanden gelöscht. Ist der Zieloperand in einem Datenregister, so erfolgt die Bitnumerierung modulo 32, wodurch die Manipulation aller Bits in einem Datenregister möglich wird. Ist der Zieloperand in einer Speicherstelle, so wird ein Byte aus dieser Stelle gelesen, die Bitoperation unter Verwendung der Bitnummer modulo 8 durchgeführt und das Byte wieder an die Speicherstelle zurückgeschrieben. Die Bitnummer kann auf zwei Arten festgelegt sein:

1. Unmittelbar: Die Bitnummer ist in einem zweiten Wort des Befehls angegeben.
2. Register: Die Bitnummer ist in einem Datenregister enthalten, das im Befehl bezeichnet wird.

Bedingungskodes:

X	N	Z	V	C
–	–	★	–	–

N nicht beeinflußt.
Z wird gesetzt, wenn das getestete Bit Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C, X nicht beeinflußt.

Befehlsformat (Bitnummer dynamisch):

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	0	0	0	Register	1	1	0	Effektive Adresse

Test a Bit and Clear

Prüfe ein Bit und lösche es

BCLR

Befehlsfelder:

Registerfeld	bezeichnet das Datenregister, das die Bitnummer enthält.
Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Befehlsformat (Bitnummer statisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	Effektive Adresse					
Bitnummer															

Befehlsfelder:

Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Bitnummerfeld	bezeichnet die Bitnummer.

BRA

Branch Always

Verzweige immer

Operation: PC + d → PC
BRA <label>

Assembler Syntax: BRA <label>

Operandenlänge: Byte, Wort

Beschreibung:

Die Programmausführung wird an der Stelle PC + Adreßdistanz fortgesetzt. Die Adreßdistanz ist ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes angibt. Der Programmzähler enthält die gegenwärtige Befehlsadresse plus 2. Wenn die 8bit-Adreßdistanz im Befehlswort Null ist, wird die 16bit-Adreßdistanz verwendet, die unmittelbar auf das Befehlswort folgt.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8bit-Adreßdistanzwert							
16bit-Adreßdistanzwert, falls 8bit-Adreßdistanzwert Null ist															

Branch Always

Verzweige immer

BRA

Befehlsfelder:

8bit-Adreßdistanzfeld	enthält ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes zwischen dem Verzweigungsbefehl und dem danach auszuführenden Befehl angibt.
16bit-Adreßdistanzfeld	gestattet einen größeren Adreßdistanzwert als 8 bit. Es wird nur verwendet, wenn der 8bit-Adreßdistanzwert Null ist.

BSET

Test a Bit and Set

Prüfe ein Bit und setze es

Operation:

~ (<Bitnummer> vom Ziel) → Z
1 → <Bitnummer> vom Ziel
BSET Dn von <ea>
BSET # <data> von <ea>

Assembler Syntax:

BSET Dn, <ea>
BSET # <data>, <ea>

Operandenlänge: Byte, Langwort

Beschreibung:

Ein einzelnes Bit des Zieloperanden wird getestet; sein Wert bestimmt das Bit Z im Bedingungskode. Nach dem Test wird dieses Bit im Zieloperanden gesetzt. Ist der Zieloperand in einem Datenregister, so erfolgt die Bitnumerierung modulo 32, wodurch die Manipulation aller Bits in einem Datenregister möglich wird. Ist der Zieloperand in einer Speicherstelle, so wird ein Byte aus dieser Stelle gelesen, die Bitoperation unter Verwendung der Bitnummer modulo 8 durchgeführt und das Byte wieder an die Speicherstelle zurückgeschrieben. Die Bitnummer kann auf zwei Arten festgelegt sein:

1. Unmittelbar: Die Bitnummer ist in einem zweiten Wort des Befehls angegeben.
2. Register: Die Bitnummer ist in einem Datenregister enthalten, das im Befehl bezeichnet wird.

Bedingungskodes:

X	N	Z	V	C
–	–	★	–	–

N nicht beeinflußt.
Z wird gesetzt, wenn das getestete Bit Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C, X nicht beeinflußt.

Befehlsformat (Bitnummer dynamisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	1	1	Effektive Adresse					

Test a Bit and Set

Prüfe ein Bit und setze es

BSET

Befehlsfelder:

Registerfeld	bezeichnet das Datenregister, das die Bitnummer enthält.
Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Befehlsformat (Bitnummer statisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	Effektive Adresse					
Bitnummer															

Befehlsfelder:

Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Bitnummerfeld	bezeichnet die Bitnummer.

BSR

Branch to Subroutine

Verzweige zur Subroutine

Operation: PC → SP@; PC + d → PC
BSR <label>

Assembler Syntax: BSR <label>

Operandenlänge: Byte, Wort

Beschreibung:

Die Adresse des auf BSR unmittelbar folgenden Befehls wird in den System-Stack geladen. Die Programmausführung wird dann an der Stelle PC + Adreßdistanz fortgesetzt. Die Adreßdistanz ist ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes angibt. Der Programmzähler enthält die gegenwärtige Befehlsadresse plus 2. Wenn die 8bit-Adreßdistanz im Befehlswort Null ist, wird die 16bit-Adreßdistanz verwendet, die unmittelbar auf das Befehlswort folgt.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8bit-Adreßdistanzwert							
16bit-Adreßdistanzwert, falls 8bit-Adreßdistanzwert Null ist															

Befehlsfelder:

8bit-Adreßdistanzfeld	enthält ein ganzzahliges Zweier-Komplement, das die relative Entfernung in Bytes zwischen dem Verzweigungsbefehl und dem danach auszuführenden Befehl angibt.
16bit-Adreßdistanzfeld	gestattet einen größeren Adreßdistanzwert als 8 bit. Es wird nur verwendet, wenn der 8bit-Adreßdistanzwert Null ist.

BTST

Test a Bit

Prüfe ein Bit

Operation:

~ (<Bitnummer> vom Ziel) → Z
BTST Dn von <ea>
BTST # <data> von <ea>

Assembler Syntax:

BTST Dn, <ea>
BTST # <data>, <ea>

Operandenlänge: Byte, Langwort

Beschreibung:

Ein einzelnes Bit des Zieloperanden wird getestet; sein Wert bestimmt das Bit Z im Bedingungskode. Ist der Zieloperand in einem Datenregister, so erfolgt die Bitnumerierung modulo 32, wodurch die Manipulation aller Bits in einem Datenregister möglich wird. Ist der Zieloperand in einer Speicherstelle, so wird ein Byte aus dieser Stelle gelesen und die Bitoperation unter Verwendung der Bitnummer modulo 8 durchgeführt. Die Bitnummer kann auf zwei Arten festgelegt sein:

1. Unmittelbar: Die Bitnummer ist in einem zweiten Wort des Befehls angegeben.
2. Register: Die Bitnummer ist in einem Datenregister enthalten, das im Befehl bezeichnet wird.

Bedingungskodes:

X	N	Z	V	C
–	–	★	–	–

N nicht beeinflußt.
Z wird gesetzt, wenn das getestete Bit Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C, X nicht beeinflußt.

Befehlsformat (Bitnummer dynamisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Register			1	0	0	Effektive Adresse					

Test a Bit

Prüfe ein Bit

BTST

Befehlsfelder:

Registerfeld	bezeichnet das Datenregister, das die Bitnummer enthält.
Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Befehlsformat (Bitnummer statisch):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	Effektive Adresse					
Bitnummer:															

Befehlsfelder:

Effektives Adreßfeld	gibt die Zieladresse an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Bitnummerfeld	bezeichnet die Bitnummer.

CHK

Check Register against Bounds

Prüfe Register auf Grenzen

Operation: Wenn Dn <0 oder Dn> (<ea>), dann TRAP
CHK Dn mit <ea>

Assembler Syntax: CHK <ea>, Dn

Operandenlänge: Wort

Beschreibung:

Das untere Wort des im Befehl angegebenen Datenregisters wird mit dem im effektiven Adreßfeld bezeichneten Operanden verglichen, dessen obere Grenze als ganzzahliges Zweier-Komplement behandelt wird. Wenn der Registerwert kleiner als Null oder größer als der Wert dieses Operanden ist, leitet der Prozessor eine Ausnahmeverarbeitung ein (Trap). Für den Ausnahmevektor des Befehls CHK wird die Vektornummer 6 (Adresse 18_{16}) erzeugt.

Bedingungskodes;

X	N	Z	V	C
–	★	U	U	U

N wird gesetzt, wenn Dn < 0; wird gelöscht, wenn Dn > (<ea>). Anderenfalls ist es nicht definiert.

Z, V, C nicht definiert.

X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	1	0	0	Register	1	1	0	Effektive Adresse

Befehlsfelder:

Registerfeld — bezeichnet das Datenregister, dessen Inhalt geprüft wird.

Effektives Adreßfeld — adressiert den Operandenwert für die obere Grenze. Es ist nur die Adressierungkategorie „Daten" zulässig.

Clear an Operand

Lösche einen Operanden

CLR

Operation: 0 → Ziel
CLR <ea>

Assembler Syntax: CLR.s <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Im Zieloperanden werden alle Bits auf Null gesetzt. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	0	1	0	0

N wird gelöscht.
Z wird gesetzt.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Länge		Effektive Adresse					

Befehlsfelder:

Längenfeld — bestimmt die Länge der Operation:
00 ≙ Byte
01 ≙ Wort
10 ≙ Langwort

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

CMP

Compare

Vergleiche

Operation: (Ziel) – (Quelle)
CMP <ea> mit Dn

Assembler Syntax: CMP.s <ea>, Dn

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird vom Zieloperanden subtrahiert, und die Bedingungskodes werden entsprechend dem Ergebnis gesetzt. Der Zieloperand bleibt ungeändert. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner als der Quellenoperand ist (negativer Übertrag). Anderenfalls wird es gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	1	1	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld — bestimmt das Ziel-Datenregister.

Operationsmodusfeld

Byte	Wort	Langwort	Operation
000	001	010	(<Dn>) – (<ea>)

Effektives Adreßfeld — bestimmt den Quellenoperanden.
Alle Adressierungskategorien sind zulässig. Hat der Operand nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

Compare Address

Vergleiche Adresse

CMPA

Operation: (Ziel) – (Quelle)
CMP <ea> mit An

Assembler Syntax: CMPA.s <ea>, An

Operandenlänge: s = (Wort, Langwort)

Beschreibung:

Der Quellenoperand wird vom Ziel-Adreßregister subtrahiert, und die Bedingungskodes werden entsprechend dem Ergebnis gesetzt. Das Adreßregister bleibt ungeändert. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner als der Quellenoperand ist (negativer Übertrag). Anderenfalls wird C gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	1	1	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld	bestimmt das Ziel-Adreßregister.
Operationsmodusfeld	bestimmt die Länge der Operation: 011 ≙ Wort. Der Quellenoperand ist durch Vorzeichenerweiterung zu einem Langwort geworden, und die Operation wird mit allen 32 Bits des Adreßregisters durchgeführt. 111 ≙ Langwort.
Effektives Adreßfeld	bestimmt den Quellenoperanden. Alle Adressierungskategorien sind zulässig.

CMPI

Compare Immediate

Vergleiche unmittelbar

Operation: (Ziel) – Unmittelbare Daten
CMP # <data> mit <ea>

Assembler Syntax: CMPI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden vom Zieloperanden subtrahiert und die Bedingungskodes entsprechend dem Ergebnis gesetzt. Der Zieloperand bleibt ungeändert.
Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner als das Datenwort ist (negativer Übertrag). Anderenfalls wird C gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamten unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

CMPM

Compare Memory

Vergleiche Speicher

Operation: (Ziel) – (Quelle)
CMP Ay@+ mit Ax@+

Assembler Syntax: CMPM.s (Ay)+, (Ax)+

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird vom Zieloperanden subtrahiert, und die Bedingungskodes werden entsprechend dem Ergebnis gesetzt. Der Zieloperand bleibt ungeändert. Die Operanden werden immer mittels der Adressierungsart „Adreßregister indirekt mit Postinkrement" adressiert, und zwar unter Verwendung der im Befehl angegebenen Adreßregister. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner als der Quellenoperand ist (negativer Übertrag). Anderenfalls wird C gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	1	1	Register Rx	1	Länge	0	0	1	Register Ry

Befehlsfelder:

Registerfeld Rx	(immer das Ziel) bezeichnet ein Adreßregister für die Adressierungsart „Adreßregister indirekt mit Post-inkrement".
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Registerfeld Ry	(immer die Quelle) bezeichnet ein Adreßregister für die Adressierungsart „Adreßregister indirekt mit Post-inkrement".

DBcc

Test Condition, Decrement and Branch

Prüfe Bedingung, dekrementiere und verzweige

Operation: Wenn $_{CC}$, dann Dn $-1 \rightarrow$ Dn;
wenn Dn $\neq -1$, dann PC + d $\rightarrow$ PC
DB_{CC} Dn, <label>

Assembler Syntax: DB_{CC} Dn, <label>

Operandenlänge: Wort

Beschreibung:

Dieser Befehl stellt eine einfache Programmschleife dar, die aus drei Parametern besteht: aus einer Bedingung, einem Datenregister und einer Adreßdistanz. Zuerst wird geprüft, ob die Abbruchbedingung „$_{CC}$" für die Schleife erfüllt ist. Falls dies zutrifft, wird der nächste Befehl durchgeführt. Ist die Abbruchbedingung „$_{CC}$" nicht erfüllt, so werden die unteren 16 Bits des angegebenen Datenregisters um 1 vermindert. Erreicht dieser Zähler den Wert -1, so führt der Prozessor den nächsten Befehl aus. Ist der Wert des Zählers dagegen $\neq -1$, so wird die Programmausführung an der Stelle fortgesetzt, die sich aus dem gegenwärtigen Programmzählerwert plus einem vorzeichenerweiterten 16bit-Adreßdistanzwert ergibt. Der neue Programmzählerwert bewegt sich in einem Rahmen von $-32\,766$ bis $+32\,769$ vom gegenwärtigen Wert. Die Bedingungen „$_{CC}$" sind folgendermaßen definiert:

CC	0100	Übertragsbit zurückgesetzt	LS	0011	niedriger oder identisch
CS	0101	Übertragsbit gesetzt	LT	1101	weniger
EQ	0111	gleich	MI	1011	negativ
F	0001	falsch	NE	0110	ungleich
GE	1100	größer oder gleich	PL	1010	positiv
GT	1110	größer	T	0000	wahr
HI	0010	höher	VC	1000	kein Überlauf
LE	1111	kleiner oder gleich	VS	1001	Überlauf

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Test Condition, Decrement and Branch

Prüfe Bedingung, dekrementiere und verzweige

DBcc

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Bedingung				1	1	0	0	1	Register		
Adreßdistanzwert															

Befehlsfelder:

Bedingungsfeld	enthält eine der in der Beschreibung angegebenen 16 Bedingungen.
Registerfeld	bestimmt das Datenregister, in dem gezählt wird.
Adreßdistanzfeld	enthält den Adreßdistanzwert in Bytes.

DIVS

Signed Divide

Division mit Vorzeichen

Operation: (Ziel) / (Quelle) → Ziel
DIVS Dn dividiert durch <ea>

Assembler Syntax: DIVS <ea>, Dn

Operandenlänge: Wort

Beschreibung:

Der Zieloperand wird durch den Quellenoperanden dividiert und mit dem Ergebnis überschrieben. Der Zieloperand besteht aus 32 Bits (Langwort), der Quellenoperand aus 16 Bits (Wort). Die Operation erfolgt unter Berücksichtigung der Vorzeichen. Das Ergebnis besteht aus 32 Bits, wobei sich

1. der Quotient im unteren Wortteil befindet (niederwertige 16 Bits),
2. der Rest im oberen Wortteil befindet (höherwertige 16 Bits).

Das Vorzeichen des Restes ist immer mit dem des Dividenden identisch, wenn nicht der Rest Null ist. Zwei spezielle Fälle können auftreten:

1. Eine Division durch Null verursacht einen Trap an die Vektornummer 5 (Adresse 14_{16}).
2. Wenn der Quellenoperand größer als der Zieloperand ist, tritt ein Überlauf ein. Dieser wird vor der Operation erkannt. In diesem Fall wird das entsprechende Bedingungsbit gesetzt, aber die Operanden bleiben unbeeinflußt.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	0

N wird gesetzt, wenn der Quotient negativ ist. Anderenfalls wird N gelöscht. Bei einem Überlauf ist N nicht definiert.
Z wird gesetzt, wenn der Quotient Null ist. Ist er ungleich Null, wird Z gelöscht. Bei einem Überlauf ist Z nicht definiert.
V wird gesetzt, wenn vor der Division ein Überlauf erkannt wird. Anderenfalls wird V gelöscht.
C wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
1	0	0	0	Register	1	1	1	Effektive Adresse

Signed Divide

Division mit Vorzeichen

DIVS

Befehlsfelder:

Registerfeld	bestimmt eines der acht Datenregister. Dieses Feld gibt immer den Zieloperanden an.
Effektives Adreßfeld	bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

DIVU

Unsigned Divide

Division ohne Vorzeichen

Operation: (Ziel) / (Quelle) → Ziel
DIVU Dn dividiert durch <ea>

Assembler Syntax: DIVU <ea>, Dn

Operandenlänge: Wort

Beschreibung:

Der Zieloperand wird durch den Quellenoperanden dividiert und mit dem Ergebnis überschrieben. Der Zieloperand besteht aus 32 Bits (Langwort), der Quellenoperand aus 16 Bits (Wort). Die Operation erfolgt ohne Vorzeichen. Das Ergebnis besteht aus 32 Bits, wobei sich

1. der Quotient im unteren Wortteil befindet (niederwertige 16 Bits),
2. der Rest im oberen Wortteil befindet (höherwertige 16 Bits).

Zwei spezielle Fälle können auftreten:

1. Eine Division durch Null verursacht einen Trap an die Vektornummer 5 (Adresse 14_{16}).
2. Wenn der Quellenoperand größer als der Zieloperand ist, tritt ein Überlauf ein. Dieser wird vor der Operation erkannt. In diesem Fall wird das entsprechende Bedingungsbit gesetzt, aber die Operanden bleiben unbeeinflußt.

Bedingungskodes:

X	N	Z	V	C
–	★	★	★	0

N wird gesetzt, wenn das höchstwertige Bit des Quotienten gesetzt ist. Anderenfalls wird N gelöscht. Bei einem Überlauf ist N nicht definiert.
Z wird gesetzt, wenn der Quotient Null ist. Ist er ungleich Null, wird Z gelöscht. Bei einem Überlauf ist Z nicht definiert.
V wird gesetzt, wenn vor der Division ein Überlauf erkannt wird. Anderenfalls wird V gelöscht.
C wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
1	0	0	0	Register	0	1	1	Effektive Adresse

Unsigned Divide

Division ohne Vorzeichen

DIVU

Befehlsfelder:

Registerfeld	bestimmt eines der acht Datenregister. Dieses Feld gibt immer den Zieloperanden an.
Effektives Adreßfeld	bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

EOR

Exclusive OR logical

Logisches exklusives ODER

Operation: (Ziel) ≢ (Quelle) → Ziel
EOR Dn zu <ea>

Assembler Syntax: EOR.s Dn, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird mit dem Zieloperanden über ein logisches exklusives ODER verknüpft und der Zieloperand mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Diese Operation ist nur mit einem Datenregister als Quellenoperand zulässig. Der Zieloperand ist im effektiven Adreßfeld festgelegt.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	1	1	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld — bestimmt eines der acht Datenregister.

Operationsmodusfeld

Byte	Wort	Langwort	Operation
100	101	110	(<ea>) ≢ (<Dx>) → <ea>

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

EORI

Exclusive OR Immediate

Exklusives ODER unmittelbar

Operation: (Ziel) ≠ unmittelbare Daten → Ziel
EORI # <data> zu <ea>

Assembler Syntax: EORI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden mit dem Zieloperanden über ein logisches exklusives ODER verknüpft; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher. Ein Zugriff zum Statusregister ist auf zwei Arten möglich:

1. Wenn die Operationslänge ein Byte beträgt, beeinflußt die Operation nur das untere Byte des Statusregisters, das die Bedingungskodes enthält.
2. Wenn die Operationslänge ein Wort beträgt, erstreckt sich die Operation auf das gesamte Statusregister. Dies ist eine privilegierte Operation, die nur in der Betriebsart „Supervisor" durchgeführt werden kann.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.
Wenn das Statusregister adressiert wird, werden die Bedingungskodes entsprechend der logischen Verknüpfung beeinflußt.

Exclusive OR Immediate

Exklusives ODER unmittelbar

EORI

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es sind nur die Adressierungskategorie „Daten ∧ Veränderbar" sowie die Adressierungsart „Unmittelbar" (ausschließlich bezogen auf das Statusregister) zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamten unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

EXG

Exchange Registers

Vertausche Daten zwischen Registern

Operation: Rx ↔ Ry
EXG Rx mit Ry

Assembler Syntax: EXG Rx, Ry

Operandengröße: Langwort

Beschreibung:

Die Inhalte von zwei Registern werden miteinander vertauscht. Dieser Austausch ist immer eine 32bit-Operation (Langwortoperation). Mit diesem Befehl lassen sich Vertauschungen durchführen zwischen:
1. Datenregistern,
2. Adreßregistern,
3. einem Datenregister und einem Adreßregister.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6 5 4 3	2 1 0
1	1	0	0	Register Rx	1	Operationsmodus	Register Ry

Befehlsfelder:

Registerfeld Rx	bestimmt — abhängig vom Modusfeld — entweder ein Daten- oder ein Adreßregister. Erfolgt der Austausch zwischen einem Daten- und einem Adreßregister, so gibt dieses Feld immer das Datenregister an.
Operationsmodusfeld	gibt an, zwischen welchen Registern der Austausch stattfindet: 01000 zwischen Datenregistern, 01001 zwischen Adreßregistern, 10001 zwischen einem Daten- und einem Adreßregister.
Registerfeld Ry	bestimmt — abhängig vom Modusfeld — entweder ein Daten- oder ein Adreßregister. Erfolgt der Austausch zwischen einem Daten- und einem Adreßregister, so gibt dieses Feld immer das Adreßregister an.

Sign Extend

Vorzeichenerweiterung

EXT

Operation: (Ziel) vorzeichenerweitert → Ziel
EXT Dn

Assembler Syntax: EXT.s Dn

Operandenlänge: s = (Wort, Langwort)

Beschreibung:

Ein Datenregister mit Vorzeichenbit wird — abhängig von der vorgegebenen Länge — durch Vorzeichenerweiterung von Byte- auf Wortlänge oder von Wort- auf Langwortlänge gebracht. Hat der Operand Wortlänge, so wird Bit 7 des angegebenen Datenregisters in die Bits 8 – 15 desselben Registers kopiert; hat er Langwortlänge, so wird Bit 15 des angegebenen Datenregisters in die Bits 16 – 31 kopiert.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8 7 6	5	4	3	2 1 0
0	1	0	0	1	0	0	Operations-modus	0	0	0	Register

Befehlsfelder:

Operationsmodusfeld — bestimmt die Länge der Vorzeichenerweiterung:
010 ≙ das untere Byte des Datenregisters wird durch Vorzeichenerweiterung zum Wort.
011 ≙ das untere Wort des Datenregisters wird durch Vorzeichenerweiterung zum Langwort.

Registerfeld — bestimmt das Datenregister, dessen Inhalt vorzeichenerweitert werden soll.

ILLEGAL

Illegal Instruction

Unzulässiger Befehl (TRAP)

Operation: PC → SSP@ − ; SR → SSP@ − ; (Vektor) → PC

Operandenlänge: nicht definiert

Beschreibung:

Dieser Befehl erzeugt die Ausnahme „Unzulässiger Befehl". Alle anderen unzulässigen Befehlskodes sind für eine zukünftige Erweiterung des Befehlsvorrats reserviert.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

Jump

Springe

JMP

Operation: Ziel → PC
JMP <ea>

Assembler Syntax: JMP <ea>

Beschreibung:

Das Programm wird bei der Adresse fortgesetzt, die durch den Befehl festgelegt ist. Die effektive Adresse gehört zur Adressierungskategorie Steuerung.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — bestimmt die Adresse des nächsten Befehls. Es ist nur die Adressierungskategorie „Steuerung" zulässig.

JSR

Jump to Subroutine

Springe zur Subroutine

Operation: PC → SP@ − ; Ziel → PC
JSR <ea>

Assembler Syntax: JSR <ea>

Beschreibung:

Die Adresse des Befehls, der dem Befehl JSR unmittelbar folgt, wird in den System-Stack gerettet. Das Programm wird dann bei der Adresse fortgesetzt, die durch den Befehl festgelegt ist.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, B, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — gibt die Adresse des nächsten Befehls an. Es ist nur die Adressierungskategorie „Steuerung" zulässig.

Load Effective Address

Lade die effektive Adresse

LEA

Operation: Ziel → An
LEA <ea> in An

Assembler Syntax: LEA <ea>, An

Operandengröße: Langwort

Beschreibung:

Die effektive Adresse wird in das angegebene Adreßregister geladen. Alle 32 Bits dieses Registers werden beeinflußt.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
0	1	0	0	Register	1	1	1	Effektive Adresse

Befehlsfelder:

Registerfeld — bezeichnet das Adreßregister, das mit der effektiven Adresse geladen werden soll.

Effektives Adreßfeld — gibt die Adresse an, die in das Adreßregister geladen werden soll. Es ist nur die Adressierungskategorie „Steuerung" zulässig.

LINK

Link and Allocate

Anbinden und Reservieren

Operation: An → SP@ − ; SP → An; SP + d → SP
LINK An, # <Adreßdistanz>

Assembler Syntax: LINK An, # <Adreßdistanz>

Operandenlänge: Keine Formatangabe

Beschreibung:

Der gegenwärtige Inhalt des angegebenen Adreßregisters wird in den Stack gerettet und belegt dort zwei Wörter. Danach wird das Adreßregister mit dem neuen Stack pointer-Inhalt geladen, und die vorzeichenerweiterte Adreßdistanz wird zum Stack pointer-Inhalt addiert.
Der Befehl LINK erlaubt es, z. B. bei verschachtelten Subroutinen oder Interruptroutinen mehrere Datenspeicherbereiche im Stack aneinanderzuhängen, von denen der jeweils zuletzt angebundene der gerade ablaufenden Routine zugeordnet ist. Diese Speicherbereiche können zur Ablage von temporären Daten oder Parametern genutzt werden. Nach Ausführung des Befehls enthält das angegebene Adreßregister die Anfangsadresse des reservierten Speicherbereiches (minus 2), während der Stack pointer A7 bzw. A7′ auf die Endadresse dieses Speicherbereiches zeigt. Der im Befehl angegebene Adreßdistanzwert darf normalerweise nur negativ sein, da sonst bereits vorhandene Stackinhalte überschrieben werden.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		
Adreßdistanzwert															

Befehlsfelder:

Registerfeld — bezeichnet das Adreßregister, das die Anfangsadresse des reservierten Stackbereiches enthält.

Adreßdistanzfeld — gibt das vorzeichenbehaftete Zweier-Komplement an, das zum Stack pointer-Inhalt addiert wird.

Logical Shift

Logische Verschiebung

LSL, LSR

Operation:

(Ziel) verschoben um <Stellenzahl> → Ziel
LSd Dy durch Dx
LSd Dy durch # <data>
LSd <ea>

Assembler Syntax:

LSd.s Dx, Dy
LSd. # <data>, Dy
LSd <ea>

Operandengröße:

s = Byte, Wort, Langwort

Beschreibung:

Die Bits des Operanden werden in der angegebenen Richtung verschoben. Das zuletzt aus dem Operanden geschobene Bit wird zum Übertragsbit. Die Anzahl der verschobenen Stellen in einem Register kann auf zwei Arten angegeben werden:

1. Unmittelbar: Die Stellenzahl ist im Befehl angegeben (1 bis 8 Stellen, wobei 8 ≙ 000).
2. Register: Die Stellenzahl ist in einem Datenregister enthalten, das im Befehl angegeben ist.

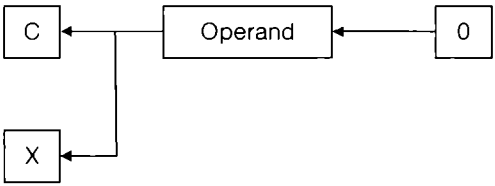
Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt einer Speicherstelle kann nur um ein Bit verschoben werden, wobei die Operandenlänge auf ein Wort beschränkt ist.

LSL, LSR

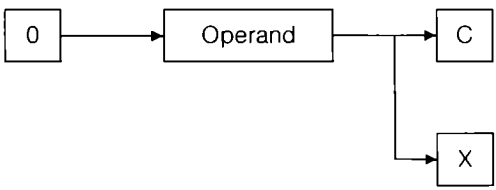
Logical Shift

Logische Verschiebung

LSL: Der Operand wird nach links um die angegebene Stellenzahl verschoben. Das aus der höchstwertigen Stelle geschobene Bit wird sowohl zum Übertrags- als auch zum Erweiterungsbit; in die niedrigstwertige Stelle wird eine Null nachgeschoben.



LSR: Der Operand wird nach rechts um die angegebene Stellenzahl verschoben. Das aus der niedrigstwertigen Stelle geschobene Bit wird sowohl zum Übertrags- als auch zum Erweiterungsbit; in die höchstwertige Stelle wird eine Null nachgeschoben.



Bedingungskodes:

X	N	Z	V	C
★	★	★	0	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gelöscht.
C wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, wird C gelöscht.
X wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, bleibt X unbeeinflußt.

Befehlsformat (Verschiebung eines Registers):

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Stellenzahl/ Register	dr	Länge	i/r	0	1	Register

Logical Shift

Logische Verschiebung

LSL, LSR

Befehlsfelder:

Stellenzahl-/Registerfeld	bestimmt die Stellenzahl oder das Register, in dem sich die Stellenzahl befindet. i/r = 0: Die Bits 9 bis 11 enthalten die Stellenzahl; die Werte 0 bis 7 entsprechen einem Bereich von 8, wobei 8 ≙ 000. i/r = 1: Die Bits 9 bis 11 geben das Datenregister an, in dem sich die Stellenzahl (modulo 64) befindet.
Bit dr	gibt die Verschieberichtung an: 0 ≙ Verschiebung nach rechts 1 ≙ Verschiebung nach links
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit i/r	i/r = 0: Die Stellenzahl ist im Befehl enthalten (unmittelbare Daten). i/r = 1: Die Stellenzahl befindet sich in dem Register, das die Bits 9 bis 11 bezeichnen.
Registerfeld	gibt das Datenregister an, dessen Inhalt verschoben werden soll.

Befehlsformat (Verschiebung in einer Speicherstelle):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	dr	1	1	Effektive Adresse					

Befehlsfelder:

Bit dr	gibt die Verschieberichtung an: 0 ≙ Verschiebung nach rechts 1 ≙ Verschiebung nach links
Effektives Adreßfeld	bestimmt den Operanden. Es ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" zulässig.

MOVE

Move Data from Source to Destination

Transportiere Daten von der Quelle zum Ziel

Operation: (Quelle) → Ziel
MOVE <ea> nach <ea>

Assembler Syntax: MOVE.s <ea>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Inhalt des Quellenoperanden wird zum Ziel transportiert. Während des Transports werden die Daten geprüft und die Bedingungskodes entsprechend gesetzt. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Quellenoperand wird nicht verändert.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht,
X nicht beeinflußt,

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Ziel						Quelle					
0	0	Länge		Register			Modus			Modus			Register		

Move Data from Source to Destination

Transportiere Daten von der Quelle zum Ziel

MOVE

Befehlsfelder:

Längenfeld	bestimmt die Länge des zu transportierenden Operanden: 01 ≙ Byte 11 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld (Ziel)	bestimmt das Ziel. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.
Effektives Adreßfeld (Quelle)	bestimmt den Quellenoperanden. Alle Adressierungskategorien sind erlaubt. Hat der Operand allerdings nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

MOVE to CCR

Move to Condition Code Register

Transportiere zum Bedingungskoderegister

Operation: (Quelle) → CCR
MOVE <ea> nach CCR

Assembler Syntax: MOVE <ea>, CCR

Operandenlänge: Wort

Beschreibung:

Der Inhalt des Quellenoperanden wird in das Bedingungskoderegister transportiert. Der Quellenoperand hat Wortlänge, aber nur das untere Byte wird verwendet.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N, Z, V, C, X: wird entsprechend dem Quellenoperanden gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
0	1	0	0	0	1	0	0	1	1	Effektive Adresse

Befehlsfelder:

Effektives Adreßfeld — bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

Move to Status Register

Transportiere zum Statusregister

MOVE to SR

Operation: (Quelle) → SR
MOVE <ea> nach SR

Assembler Syntax: MOVE <ea>, SR

Operandenlänge: Wort

Beschreibung:

Der Inhalt des Quellenoperanden wird zum Statusregister transportiert. Der Quellenoperand hat Wortlänge, und alle Bits des Statusregisters werden beeinflußt.

Zur Beachtung: Dies ist ein privilegierter Befehl, der nur in der Betriebsart „Supervisor" zugelassen ist.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N, Z, V, C, X } wird entsprechend dem Quellenoperanden gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

MOVE from SR

Move from the Status Register

Transportiere vom Statusregister

Operation: SR → Ziel
MOVE SR nach <ea>

Assembler Syntax: MOVE SR, <ea>

Operandenlänge: Wort

Beschreibung:

Der Inhalt des Statusregisters wird zum Ziel transportiert. Der Operand hat Wortlänge.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Move User Stack Pointer

Transportiere den Inhalt des Anwender-Stack pointers

MOVE USP

Operation:

USP → An
An → USP
MOVE USP zum An
MOVE An zum USP

Assembler Syntax:

MOVE USP, An
MOVE An, USP

Operandenlänge: Langwort

Beschreibung:

Der Inhalt des Anwender-Stack pointers A7 wird in das angegebene Adreßregister transportiert bzw. vom Adreßregister in den Anwender-Stack. Der jeweilige Quellenoperand bleibt unverändert.

Zur Beachtung: Dies ist ein privilegierter Befehl, der nur in der Betriebsart „Supervisor" zugelassen ist.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	Register		

Befehlsfelder:

Bit dr — gibt die Transportrichtung an:
0 ≙ Transport des Adreßregister-Inhalts zum USP
1 ≙ Transport des USP-Inhalts zum Adreßregister

Registerfeld — gibt das Adreßregister an, zu dem oder von dem der USP-Inhalt transportiert werden soll.

MOVEA

Move Address

Transportiere Adresse

Operation: (Quelle) → Ziel
MOVE <ea> zu An

Assembler Syntax: MOVEA.s <ea>, An

Operandenlänge: s = (Wort, Langwort)

Beschreibung:

Der Inhalt der Quelle wird zum Ziel-Adreßregister transportiert. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Ziel						Quelle					
0	0	Länge		Register			0	0	1	Modus			Register		

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 11 ≙ Wort. Der Quellenoperand wird durch Vorzeichenerweiterung zum Langwort; alle 32 Bits werden in das Adreßregister geladen. 10 ≙ Langwort
Effektives Adreßfeld (Ziel)	bestimmt das Ziel-Adreßregister.
Effektives Adreßfeld (Quelle)	bestimmt den Quellenoperanden. Alle Adressierungskategorien sind zulässig.

Move Multiple Registers

Transportiere mehrere Register

MOVEM

Operation:

Mehrere Register → Ziel
(Quelle) → mehrere Register
MOVEM <Registerliste> zum <ea>
MOVEM <ea> zur <Registerliste>

Assembler Syntax:

MOVEM.s <Registerliste>, <ea>
MOVEM.s <ea>, <Registerliste>

Operandenlänge:

s = (Wort, Langwort)

Beschreibung:

Der Inhalt der angegebenen Register wird zu aufeinanderfolgenden Speicherstellen transportiert, oder der Inhalt aufeinanderfolgender Speicherstellen wird in die angegebenen Register geladen. In jedem Fall wird mit derjenigen Speicherstelle begonnen, die durch die effektive Adresse festgelegt ist. Der Inhalt eines Registers wird dann transportiert, wenn das diesem Register im Maskenfeld zugeordnete Bit gesetzt ist. Im Befehl ist auch vorgegeben, ob der gesamte Register-Inhalt (ein Langwort) oder nur das untere Wort dieses Registers zu transportieren ist. Soll ein Wort in ein Register transportiert werden, so wird es auf 32 bit vorzeichenerweitert und somit als Langwort in das betreffende Register geladen.
Für den Befehl MOVEM sind die Adressierungskategorie „Steuerung" sowie die Adressierungsarten „Adreßregister indirekt mit Postinkrement" und „Adreßregister indirekt mit Predekrement" zugelassen.
Bewegt sich die effektive Adresse in der Adressierungskategorie „Steuerung", so beginnt der Transport vom Speicher in die Register bei der angegebenen Adresse und setzt sich mit den folgenden höheren Adressen fort. Die Reihenfolge für den Transport ist: zuerst Datenregister von 0 bis 7, danach Adreßregister von 0 bis 7.
Beim Transport von den Registern in den Speicher ist nur die Adressierungskategorie „Steuerung ∧ Veränderbar" zulässig.
Bei der Adressierungsart „Adreßregister indirekt mit Predekrement" ist ebenfalls nur der Transport von den Registern zum Speicher zulässig. Die Speicherung der Registerinhalte beginnt bei der angegebenen Adresse minus 2 und setzt sich mit den folgenden niedrigeren Adressen fort. Die Reihenfolge der Speicherung ist: zuerst Adreßregister von 7 bis 0, danach Datenregister von 7 bis 0. Das dekrementierte Adreßregister wird auf den neuesten Stand gebracht und enthält dann die Adresse des zuletzt gespeicherten Wortes. Bei der Adressierungsart „Adreßregister indirekt mit Postinkrement" ist nur der Transport vom Speicher zu den Registern zulässig.
Das Laden der Register beginnt bei der angegebenen Adresse und setzt sich mit den folgenden höheren Adressen fort. Die Reihenfolge beim Laden ist dieselbe wie bei der Adressierungskategorie „Steuerung". Das inkrementierte Adreßregister wird auf den neuesten Stand gebracht und enthält dann die Adresse des zuletzt geladenen Wortes plus 2.

MOVEM

Move Multiple Registers

Transportiere mehrere Register

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	sz	Effektive Adresse					
Registerlisten-Maske															

Move Multiple Registers

Transportiere mehrere Register

MOVEM

Befehlsfelder:

Bit dr	gibt die Transporteinrichtung an: 0 ≙ Register zum Speicher 1 ≙ Speicher zum Register
Bit sz	bestimmt die Länge des zu transportierenden Register-Inhalts: 0 ≙ Transport eines Wortes 1 ≙ Transport eines Langwortes
Effektives Adreßfeld	enthält die Adresse des Speichers, zu dem oder von dem die Register-Inhalte transportiert werden sollen. Beim Transport von den Registern zum Speicher sind nur die Adressierungskategorie „Steuerung ∧ Veränderbar" sowie die Adressierungsart „Adreßregister indirekt mit Predekrement" zulässig. Beim Transport vom Speicher zu den Registern sind nur die Adressierungskategorie „Steuerung" sowie die Adressierungsart „Adreßregister indirekt mit Postinkrement" zulässig.
Registerlisten-Maskenfeld	bestimmt, welche Register-Inhalte transportiert werden sollen. Das niedrigstwertige Bit entspricht dem ersten Register-Inhalt, das höchstwertige dem letzten Register-Inhalt, der transportiert werden soll. Somit hat die Maske für alle Adressierungsarten und -kategorien folgenden Aufbau:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

mit Ausnahme der Adressierungsart „Adreßregister indirekt mit Predekrement":

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

MOVEP

Move Peripheral Data

Transportiere periphere Daten

Operation: (Quelle) → Ziel
MOVEP Dx zu Ay@(d)
MOVEP Ay@(d) zu Dx

Assembler Syntax: MOVEP Dx, d(Ay)
MOVEP d(Ay), Dx

Operandenlänge: Wort, Langwort

Beschreibung:

Der Datentransport findet zwischen einem Datenregister und alternierenden Bytes des Speichers statt, wobei an der angegebenen Stelle begonnen und um 2 inkrementiert wird. Der Inhalt des verwendeten Adreßregisters wird nicht verändert. Das höchstwertige Byte des Datenregisters wird zuerst, das niedrigstwertige Byte zuletzt transportiert. Die Angabe der Speicheradresse erfolgt unter Verwendung der Adressierungsart „Adreßregister indirekt mit Adreßdistanz". Bei einer geradzahligen Adresse werden alle Transporte auf der oberen Hälfte des Datenbusses, bei einer ungeradzahligen auf der unteren Hälfte des Datenbusses durchgeführt.

Anmerkung: Der Befehl MOVEP vereinfacht den Datentransport von oder zu Peripherieschaltungen mit 8bit-Datenbus.

Beispiel 1:
Transport eines Langwortes zu oder von einer geraden Adresse.

Byte-Organisation im Register

31 … 24	23 … 16	15 … 8	7 … 0
Byte 3	Byte 2	Byte 1	Byte 0

Byte-Organisation im Speicher

Adresse	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
n	Byte 3	
n + 2	Byte 2	
n + 4	Byte 1	
n + 6	Byte 0	

Beispiel 2:
Transport eines Wortes zu oder von einer ungeraden Adresse.

Move Peripheral Data

Transportiere periphere Daten

MOVEP

Byte-Organisation im Register

31 … 24	23 … 16	15 … 8	7 … 0
		Byte 1	Byte 0

Byte-Organisation im Speicher

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	Adresse
	Byte 1	n + 1
	Byte 0	n + 3

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5	4	3	2 1 0
0	0	0	0	Daten-register	Operations-modus	0	0	1	Adreß-register
Adreßdistanzwert									

Befehlsfelder:

Datenregisterfeld	bestimmt das Datenregister, zu oder von dem Daten transportiert werden sollen.
Operationsmodusfeld	bestimmt die Richtung und die Länge der Operation: 100 ≙ Transport eines Wortes vom Speicher zum Register 101 ≙ Transport eines Langwortes vom Speicher zum Register 110 ≙ Transport eines Wortes vom Register zum Speicher 111 ≙ Transport eines Langwortes vom Register zum Speicher
Adreßregisterfeld	bestimmt das Adreßregister, das in der Adressierungsart „Adreßregister indirekt mit Adreßdistanz" arbeitet.
Adreßdistanzfeld	bestimmt den Adreßdistanzwert, der bei der Berechnung der Operandenadresse verwendet wird.

MOVEQ

Move Quick

Transportiere schnell

Operation: Unmittelbare Daten → Ziel
MOVEQ # <data> zu Dn

Assembler Syntax: MOVEQ # <data>, Dn

Operandenlänge: Langwort

Beschreibung:

Die im Befehlswort enthaltenen unmittelbaren Daten (8 Bits) werden auf 32 Bits vorzeichenerweitert und in ein Datenregister übertragen.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6 5 4 3 2 1 0
0	1	1	1	Register	0	Daten

Befehlsfelder:

Registerfeld — bestimmt das Datenregister, das geladen werden soll.
Datenfeld — enthält die 8 Datenbits, die durch Vorzeichenerweiterung zu einem Langwort (32 Bits) werden sollen.

Signed Multiply

Multiplikation mit Vorzeichen

MULS

Operation: (Ziel) × (Quelle) → Ziel
MULS Dn mit <ea>

Assembler Syntax: MULS <ea>, Dn

Operandenlänge: Wort

Beschreibung:

Zwei 16bit-Operanden mit Vorzeichen werden miteinander multipliziert, was zu einem 32bit-Ergebnis mit Vorzeichen führt. Nur das untere Wort des Registeroperanden wird verwendet, das obere bleibt ungenutzt. Alle 32 Bits des Ergebnisses werden im Ziel-Datenregister abgelegt.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
1	1	0	0	Register	1	1	1	Effektive Adresse

Befehlsfelder:

Registerfeld — bestimmt eines der Datenregister. Dieses Feld gibt immer den Zieloperanden an.

Effektives Adreßfeld — bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

MULU

Unsigned Multiply

Multiplikation ohne Vorzeichen

Operation: (Ziel) × (Quelle) → Ziel
MULU Dn mit <ea>

Assembler Syntax: MULU <ea>, Dn

Operandenlänge: Wort

Beschreibung:

Zwei 16bit-Operanden ohne Vorzeichen werden miteinander multipliziert, was zu einem 32bit-Ergebnis ohne Vorzeichen führt. Nur das untere Wort des Registeroperanden wird verwendet, das obere bleibt ungenutzt. Alle 32 Bits des Ergebnisses werden im Ziel-Datenregister abgelegt.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5 4 3 2 1 0
1	1	0	0	Register	0	1	1	Effektive Adresse

Befehlsfelder:

Registerfeld — bestimmt eines der Datenregister. Dieses Feld gibt immer den Zieloperanden an.

Effektives Adreßfeld — bestimmt den Quellenoperanden. Es ist nur die Adressierungskategorie „Daten" zulässig.

Negate Decimal with Extend

Negiere dezimal mit Erweiterungsbit

NBCD

Operation: $0 - (\text{Ziel})_{10} - X \rightarrow \text{Ziel}$
NBCD <ea>

Assembler Syntax: NBCD <ea>

Operandenlänge: Byte

Beschreibung:

Der Zieloperand und das Erweiterungsbit werden von Null subtrahiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operation wird mit binär kodierten Dezimalziffern durchgeführt. Dieser Befehl ergibt das Zehner-Komplement des Operanden, wenn das Erweiterungsbit gelöscht ist, und das Neuner-Komplement, wenn das Erweiterungsbit gesetzt ist. Die Operation kann nur byteweise durchgeführt werden.

Anmerkung: Der Befehl NBCD wird vorzugsweise in arithmetischen Operationen mit mehreren BCD-Ziffern verwendet. Er erlaubt ein einfaches Berechnen von negativen Absolutzahlen.

Bedingungskodes:

X	N	Z	V	C
★	U	★	U	★

N nicht definiert.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V nicht definiert.
C wird gesetzt, wenn ein negativer (dezimaler) Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

NEG

Negate

Negiere

Operation: 0 – (Ziel) → Ziel
NEG <ea>

Assembler Syntax: NEG.s <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Zieloperand wird von Null subtrahiert und mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein negativer Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Länge		Effektive Adresse					

Befehlsfelder:

Längenfeld — bestimmt die Länge der Operation:
00 ≙ Byte
01 ≙ Wort
10 ≙ Langwort

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Negate with Extend

Negiere mit Erweiterungsbit

NEGX

Operation: 0 – (Ziel) – X → Ziel
NEGX <ea>

Assembler Syntax: NEGX.s <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Zieloperand und das Erweiterungsbit werden von Null subtrahiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Anmerkung: Der Befehl NEGX wird vorzugsweise bei arithmetischen Operationen mit mehrfacher Genauigkeit verwendet.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein negativer Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	Länge		Effektive Adresse					

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

NOP

No Operation

Keine Operation

Operation: Keine

Assembler Syntax: NOP

Operandenlänge: Keine Formatangabe

Beschreibung:

Es wird keine Operation durchgeführt. Der Prozessorzustand wird — mit Ausnahme des Programmzählers — nicht verändert. Die Programmausführung wird mit dem auf NOP folgenden Befehl fortgesetzt.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

Logical Complement

Einer-Komplement

NOT

Operation: ~ (Ziel) → Ziel
NOT <ea>

Assembler Syntax: NOT.s <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Vom Zieloperanden wird das Einer-Komplement gebildet und der Zieloperand dann mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Länge		Effektive Adresse					

Befehlsfelder:

Längenfeld — bestimmt die Länge der Operation:
00 ≙ Byte
01 ≙ Wort
10 ≙ Langwort

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

OR

Inclusive OR Logical

Logisches ODER

Operation:

(Ziel) V (Quelle) → Ziel
OR <ea> zu Dn
OR Dn zu <ea>

Assembler Syntax:

OR.s <ea>, Dn
OR.s Dn, <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird mit dem Zieloperanden über ein logisches ODER verknüpft und der Zieloperand mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt eines Adreßregisters darf nicht als Operand verwendet werden.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	0	0	Register	Operations-modus	Effektive Adresse

Inclusive OR Logical

Logisches ODER

OR

Befehlsfelder:

Registerfeld — bestimmt eines der acht Datenregister.

Operationsmodusfeld

Byte	Wort	Langwort	Operation
000	001	010	(<Dn>) V (<ea>) → <Dn>
100	101	110	(<ea>) V (<Dn>) → <ea>

Effektives Adreßfeld — bestimmt die Adressierungsart:

a) Wenn dieses Feld den Quellenoperanden adressiert, ist nur die Adressierungskategorie „Daten" zulässig.

b) Wenn dieses Feld den Zieloperanden adressiert, ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

ORI

Inclusive OR Immediate

Logisches ODER unmittelbar

Operation: (Ziel) V unmittelbare Daten → Ziel
OR # <data> zu <ea>

Assembler Syntax: ORI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden mit dem Zieloperanden über ein logisches ODER verknüpft; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher. Ein Zugriff zum Statusregister ist auf zwei Arten möglich:

1. Wenn die Operandenlänge ein Byte beträgt, beeinflußt die Operation nur das untere Byte des Statusregisters, das die Bedingungskodes enthält.
2. Wenn die Operandenlänge ein Wort beträgt, erstreckt sich die Operation auf das gesamte Statusregister. Dies ist eine privilegierte Operation, die nur in der Betriebsart „Supervisor" verwendet werden darf.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Wenn das Statusregister adressiert wird, werden die Bedingungskodes entsprechend der logischen Verknüpfung beeinflußt.

Inclusive OR Immediate

Logisches ODER unmittelbar

ORI

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es sind nur die Adressierungskategorie „Daten ∧ Veränderbar" sowie die Adressierungsart „Unmittelbar" (ausschließlich bezogen auf das Statusregister) zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamten unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

PEA

Push Effective Address

Ablegen der effektiven Adresse

Operation: Ziel → SP@ −
PEA <ea>

Assembler Syntax: PEA <ea>

Operandenlänge: Langwort

Beschreibung:

Die effektive Adresse wird berechnet und im Stack abgelegt. Es handelt sich um eine Langwort-Adresse. Der Wert des Stack pointers hat sich nach der Operation um 4 vermindert.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — gibt die Adresse an, die im Stack abgelegt werden soll. Es ist nur die Adressierungskategorie „Steuerung" zulässig.

Reset External Devices

Setze externe Einheiten zurück

RESET

Operation: Keine

Assembler Syntax: RESET

Operandenlänge: Keine Formatangabe

Beschreibung:

Die RESET-Leitung am SC 68000 wird für die Dauer von 124 Taktimpulsen aktiviert und setzt alle externen Einheiten zurück. Der Prozessorzustand wird — mit Ausnahme des Programmzählers — nicht verändert. Die Programmausführung wird mit dem nächsten Befehl fortgesetzt.

Zur Beachtung: Dies ist ein privilegierter Befehl, der nur in der Betriebsart „Supervisor" zugelassen ist.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X: nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

ROL, ROR

Rotate without Extend

Rundverschiebung ohne Erweiterungsbit

Operation:

(Ziel) rundgeschoben um <Stellenzahl> → Ziel
ROd Dy durch Dx
ROd Dy durch # <data>
ROd <ea>

Assembler Syntax:

ROd.s Dx, Dy
ROd.s # <data>, Dy
ROd <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

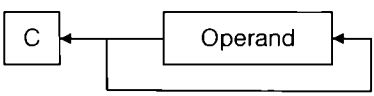
Beschreibung:

Die Bits des Operanden werden in der angegebenen Richtung rundgeschoben. Das Erweiterungsbit ist in die Rundverschiebung nicht einbezogen. Die Anzahl der rundgeschobenen Stellen in einem Register kann auf zwei Arten angegeben werden.

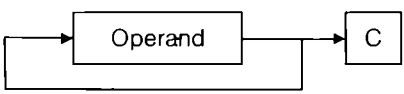
1. Unmittelbar: Die Stellenzahl ist im Befehl angegeben (1 bis 8 Stellen, wobei 8 ≙ 000).
2. Register: Die Stellenzahl ist in einem Datenregister enthalten, das im Befehl angegeben ist.

Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt einer Speicherstelle kann nur um ein Bit rundgeschoben werden, wobei die Operandenlänge auf ein Wort beschränkt ist.

ROL: Der Operand wird nach links um die angegebene Stellenzahl rundgeschoben. Das Bit aus der höchstwertigen Stelle wird sowohl zum Übertragsbit als auch zurück zur niedrigstwertigen Stelle des Operanden geschoben. Das Erweiterungsbit wird weder geändert noch verwendet.



ROR: Der Operand wird nach rechts um die angegebene Stellenzahl rundgeschoben. Das Bit aus der niedrigstwertigen Stelle wird sowohl zum Übertragsbit als auch zurück zur höchstwertigen Stelle des Operanden geschoben. Das Erweiterungsbit wird weder geändert noch verwendet.



Rotate without Extend

Rundverschiebung ohne Erweiterungsbit

ROL, ROR

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	★

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gelöscht.
C wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, wird C gelöscht.
X nicht beeinflußt.

Befehlsformat (Rundschieben eines Registers):

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Stellenzahl/ Register	dr	Länge	i/r	1	1	Register

Befehlsfelder:

Stellenzahl/Registerfeld	bestimmt die Stellenzahl oder das Register, in dem sich die Stellenzahl befindet. i/r = 0: Die Bits 9 bis 11 enthalten die Stellenzahl; die Werte 0 bis 7 entsprechen einem Bereich von 8, wobei 8 ≙ 000. i/r = 1: Die Bits 9 bis 11 geben das Datenregister an, in dem sich die Stellenzahl (modulo 64) befindet.
Bit dr	gibt die Rundschieberichtung an: 0 ≙ rundschieben nach rechts 1 ≙ rundschieben nach links
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit i/r	i/r = 0: Stellenzahl ist im Befehl enthalten (unmittelbare Daten). i/r = 1: Stellenzahl befindet sich in dem Register, das die Bits 9 bis 11 bezeichnen.
Registerfeld	gibt das Datenregister an, dessen Inhalt rundgeschoben werden soll.

ROL, ROR

Rotate without Extend

Rundverschiebung ohne Erweiterungsbit

Befehlsformat (Rundschieben einer Speicherstelle):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	dr	1	1	Effektive Adresse					

Befehlsfelder:

Bit dr	gibt die Rundschieberichtung an: 0 ≙ rundschieben nach rechts 1 ≙ rundschieben nach links
Effektives Adreßfeld	bestimmt den Operanden, der rundgeschoben werden soll. Es ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

Rotate with Extend

Rundverschiebung mit Erweiterungsbit

ROXL, ROXR

Operation:

(Ziel) rundgeschoben um <Stellenzahl> → Ziel
ROXd Dy durch Dx
ROXd Dy durch # <data>
ROXd <ea>

Assembler Syntax:

ROXd.s Dx, Dy
ROXd.s # <data>, Dy
ROXd <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

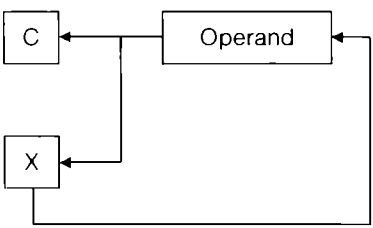
Beschreibung:

Die Bits des Operanden werden in der angegebenen Richtung rundgeschoben. Das Erweiterungsbit ist in die Rundverschiebung einbezogen. Die Anzahl der rundgeschobenen Stellen in einem Register kann auf zwei Arten angegeben werden.

1. Unmittelbar: Die Stellenzahl ist im Befehl angegeben (1 bis 8 Stellen, wobei 8 ≙ 000).
2. Register: Die Stellenzahl ist in einem Datenregister enthalten, das im Befehl angegeben ist.

Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Der Inhalt einer Speicherstelle kann nur um ein Bit rundgeschoben werden, wobei die Operandenlänge auf ein Wort beschränkt ist.

ROXL: Der Operand wird nach links um die angegebene Stellenzahl rundgeschoben. Das Bit aus der höchstwertigen Stelle wird sowohl zum Übertrags- als auch zum Erweiterungsbit geschoben; der vorherige Wert des Erweiterungsbits wird zur niedrigstwertigen Stelle des Operanden geschoben.

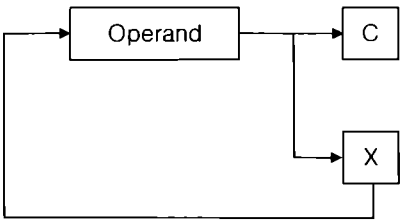


ROXL, ROXR

Rotate with Extend

Rundverschiebung mit Erweiterungsbit

ROXR: Der Operand wird nach rechts um die angegebene Stellenzahl rundgeschoben. Das Bit aus der niedrigstwertigen Stelle wird sowohl zum Übertrags- als auch zum Erweiterungsbit geschoben; der vorherige Wert des Erweiterungsbits wird zur höchstwertigen Stelle des Operanden geschoben.



Bedingungskodes:

X	N	Z	V	C
★	★	★	0	★

N wird gesetzt, wenn das höchstwertige Bit des Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gelöscht.
C wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, wird C entsprechend dem Wert des Erweiterungsbits gesetzt.
X wird entsprechend dem letzten aus dem Operanden geschobenen Bit gesetzt. Beträgt die Stellenzahl Null, wird X nicht beeinflußt.

Befehlsformat (Rundschieben eines Registers):

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	Stellenzahl/ Register	dr	Länge	i/r	1	0	Register

Rotate with Extend

Rundverschiebung mit Erweiterungsbit

ROXL, ROXR

Befehlsfelder:

Stellenzahl-/Registerfeld	bestimmt die Stellenzahl oder das Register, in dem sich die Stellenzahl befindet. i/r = 0: Die Bits 9 bis 11 enthalten die Stellenzahl; die Werte 0 bis 7 entsprechen einem Bereich von 8, wobei 8 ≙ 000). i/r = 1: Die Bits 9 bis 11 geben das Datenregister an, in dem sich die Stellenzahl (modulo 64) befindet.
Bit dr	gibt die Rundschieberichtung an: 0 ≙ rundschieben nach rechts 1 ≙ rundschieben nach links
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit i/r	i/r = 0: Stellenzahl ist im Befehl enthalten (unmittelbare Daten). i/r = 1: Stellenzahl befindet sich in dem Register, das die Bits 9 bis 11 bezeichnen.
Registerfeld	gibt das Datenregister an, dessen Inhalt rundgeschoben werden soll.

Befehlsformat (Rundschieben eines Speichers):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	dr	1	1	Effektive Adresse					

Befehlsfelder:

Bit dr	gibt die Rundschieberichtung an: 0 ≙ rundschieben nach rechts 1 ≙ rundschieben nach links
Effektives Adreßfeld	bestimmt den Operanden, der rundgeschoben werden soll. Es ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

RTE

Return from Exception

Springe von Ausnahme zurück

Operation: SP@+ → SR; SP@+ → PC
RTE

Assembler Syntax: RTE

Operandenlänge: Keine Formatangabe

Beschreibung:

Der Inhalt des Statusregisters und des Programmzählers werden aus dem System-Stack zurückgeholt. Der vorherige Stand des Statusregisters und des Programmzählers gehen verloren. Alle Bits im Statusregister werden beeinflußt.

Zur Beachtung: Dies ist ein privilegierter Befehl, der nur in der Betriebsart „Supervisor" zugelassen ist.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N, Z, V, C, X } wird entsprechend dem Inhalt des Wortes im Stack gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

Return and Restore Condition Codes

Springe zurück und lade Bedingungs-kodes

RTR

Operation: SP@+ → CC; SP@+ → PC
RTR

Assembler Syntax: RTR

Operandenlänge: Keine Formatangabe

Beschreibung:

Der Inhalt der Bedingungskodes und des Programmzählers werden aus dem System-Stack zurückgeholt. Der vorherige Stand der Bedingungskodes und des Programmzählers gehen verloren. Der Supervisor-Teil des Statusregisters bleibt unbeeinflußt.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N, Z, V, C, X } wird entsprechend dem Inhalt des Wortes im Stack gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS

Return from Subroutine

Springe zurück von der Subroutine

Operation: SP@+ → PC
RTS

Assembler Syntax: RTS

Operandenlänge: Keine Formatangabe

Beschreibung:

Der Inhalt des Programmzählers wird aus dem System-Stack zurückgeholt. Der vorherige Stand des Programmzählers geht verloren.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD

Subtract Decimal with Extend

Subtrahiere dezimal mit Erweiterungsbit

Operation: $(\text{Ziel})_{10} - (\text{Quelle})_{10} - X \rightarrow \text{Ziel}$
SBCD Dy von Dx
SBCD Ay@− von Ax@−

Assembler Syntax: SBCD Dy, Dx
SBCD −(Ay), −(Ax)

Operandenlänge: Byte

Beschreibung:

Der Quellenoperand und das Erweiterungsbit X werden vom Zieloperanden subtrahiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Subtraktion wird mit binär kodierten Dezimalziffern durchgeführt. Für die Adressierung der Operanden gibt es zwei Möglichkeiten:

1. Von Datenregister zu Datenregister. Die im Befehl angegebenen Datenregister enthalten die Operanden.
2. Von Speicher zu Speicher. Die Operanden werden in der Adressierungsart „Adreßregister indirekt mit Predekrement" unter Verwendung der im Befehl angegebenen Adreßregister adressiert.

Der Befehl wird nur in der Operandenlänge „Byte" ausgeführt.

Bedingungskodes:

X	N	Z	V	C
★	U	★	U	★

N nicht definiert.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V nicht definiert.
C wird gesetzt, wenn ein negativer (dezimaler) Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	0	0	0	Register Rx	1	0	0	0	0	R/M	Register Ry

Subtract Decimal with Extend

Subtrahiere dezimal mit Erweiterungsbit

SBCD

Befehlsfelder:

Registerfeld Rx	bezeichnet das Zielregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")
Bit R/M	bezeichnet die Adressierungsart des Operanden: 0 ≙ Operation erfolgt von Datenregister zu Datenregister 1 ≙ Operation erfolgt von Speicher zu Speicher
Registerfeld Ry	bezeichnet das Quellenregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")

Scc

Set According to Condition

Setze Byte bedingt

Operation: Wenn $_{CC}$, dann Einsen → Ziel; sonst Nullen → Ziel
S_{CC} <ea>

Assembler Syntax: S_{CC} <ea>

Operandenlänge: Byte

Beschreibung:

Die angegebene Bedingung wird geprüft. Ist sie erfüllt, wird das durch die effektive Adresse festgelegte Byte auf FF_{16} gesetzt; ist sie nicht erfüllt, wird es auf 00_{16} gesetzt. Die Bedingungen „CC" sind folgendermaßen definiert:

CC	0100	Übertragsbit zurückgesetzt	LS	0011	niedriger oder identisch
CS	0101	Übertragsbit gesetzt	LT	1101	weniger
EQ	0111	gleich	MI	1011	negativ
F	0001	falsch	NE	0110	ungleich
GE	1100	größer oder gleich	PL	1010	positiv
GT	1110	größer	T	0000	wahr
HI	0010	höher	VC	1000	kein Überlauf
LE	1111	kleiner oder gleich	VS	1001	Überlauf

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Bedingung				1	1	Effektive Adresse					

Befehlsfelder:

Bedingungsfeld — enthält eine der 16 Bedingungen.

Effektives Adreßfeld — gibt die Zieladresse für das resultierende Byte an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Load Status Register and Stop

Lade das Statusregister und stoppe

STOP

Operation: Unmittelbare Daten → SR; STOP
STOP #xxx

Assembler Syntax: STOP #xxx

Operandenlänge: Keine Formatangabe

Beschreibung:

Die unmittelbaren Daten des Operanden werden in das gesamte Statusregister transportiert, der Programmzähler rückt auf den nächsten Befehl weiter, und der Prozessor stoppt. Befehle werden erst dann wieder ausgeführt, wenn eine der Ausnahmen Trace, Interrupt oder Rücksetzen auftritt. Die Ausnahme „Trace" findet sofort statt, wenn der Trace-Modus eingeschaltet ist (Statusbit T = 1) und der Befehl „STOP" ausgeführt wurde. Wenn ein Interrupt angefordert wird, dessen Priorität höher ist als die augenblickliche Ausnahme-Priorität, so erfolgt die Ausnahme „Interrupt"; anderenfalls bleibt der Interrupt wirkungslos. Ist das dem Statusbit S entsprechende Bit in den unmittelbaren Daten zurückgesetzt, so verursacht die Ausführung dieses Befehls eine Privilegverletzung. Durch externes Rücksetzen wird stets die Ausnahme Rücksetzen eingeleitet.
Zur Beachtung: Dies ist ein privilegierter Befehl, der nur in der Betriebsart „Supervisor" zugelassen ist.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N, Z, V, C, X } wird entsprechend den unmittelbaren Daten im Operanden gesetzt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
Unmittelbare Daten															

Befehlsfelder:

Feld mit unmittelbaren Daten — gibt die Daten an, die in das Statusregister geladen werden sollen.

SUB

Subtract Binary

Subtrahiere binär

Operation:

(Ziel) – (Quelle) → Ziel
SUB <ea> von Dn
SUB Dn von <ea>

Assembler Syntax:

SUB.s <ea>, Dn
SUB.s Dn, <ea>

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand wird vom Zieloperanden subtrahiert und der Zieloperand mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Dort wird auch angegeben, wo Quelle und Ziel der Operanden sind.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner als der Quellenoperand ist (negativer Übertrag). Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	0	1	Register	Operations-modus	Effektive Adresse

Subtract Binary

Subtrahiere binär

SUB

Befehlsfelder:

Registerfeld — bestimmt eines der acht Datenregister.

Operationsmodusfeld

Byte	Wort	Langwort	Operation
000	001	010	(<Dn>) – (<ea>) → <Dn>
100	101	110	(<ea>) – (<Dn>) → <ea>

Effektives Adreßfeld — bestimmt die Adressierungsart:

a) Wenn dieses Feld den Quellenoperanden adressiert, sind alle Adressierungskategorien erlaubt. Hat der Operand allerdings nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

b) Wenn dieses Feld den Zieloperanden adressiert, ist nur die Adressierungskategorie „Speicher ∧ Veränderbar" (s. Tabelle 1) zulässig.

SUBA

Subtract Address

Subtrahiere Adresse

Operation: (Ziel) – (Quelle) → Ziel
SUB <ea> von An

Assembler Syntax: SUBA.s <ea> An

Operandenlänge: s = (Wort, Langwort)

Beschreibung:

Der Quellenoperand wird vom Zieladreßregister subtrahiert und das Ergebnis im Adreßregister abgelegt. Die Operandenlänge beträgt entsprechend der Angabe im Modusfeld des Befehls entweder ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11 10 9	8 7 6	5 4 3 2 1 0
1	0	0	1	Register	Operations-modus	Effektive Adresse

Befehlsfelder:

Registerfeld	bestimmt eines der acht Adreßregister. Dies ist immer das Zielregister.
Operationsmodusfeld	bestimmt die Länge der Operation: 011 ≙ Wortoperation. Der Quellenoperand wird durch Vorzeichenerweiterung zu einem Langwort. Die Operation erfolgt im Adreßregister unter Verwendung aller 32 Bits. 111 ≙ Langwortoperation.
Effektives Adreßfeld	bestimmt den Quellenoperanden. Alle Adressierungskategorien sind zulässig.

SUBI

Subtract Immediate

Subtrahiere unmittelbar

Operation: (Ziel) – unmittelbare Daten → Ziel
SUB # <data> von <ea>

Assembler Syntax: SUBI.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden vom Zieloperanden subtrahiert; der Zieloperand wird mit dem Ergebnis überschrieben. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Die unmittelbaren Daten sind das nächste Wort bzw. die nächsten beiden Wörter nach dem Befehlskode im Programmspeicher.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner ist als das Datenwort (negativer Übertrag). Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	Länge		Effektive Adresse					
Datenwort (16 bit)								Datenbyte (8 bit)							
Datenlangwort (32 bit, einschl. d. vorhergehenden Wortes)															

Befehlsfelder:

Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" (s. Tabelle 1) zulässig.
Feld mit unmittelbaren Daten	folgt dem Befehlswort unmittelbar. Hat das Längenfeld den Wert 00, so befinden sich die Daten im unteren Byte des unmittelbar folgenden Wortes. Hat das Längenfeld den Wert 01, so befinden sich die Daten im gesamten unmittelbar folgenden Wort. Hat das Längenfeld den Wert 10, so befinden sich die Daten in den beiden unmittelbar folgenden Wörtern.

SUBQ

Subtract Quick

Subtrahiere schnell

Operation: (Ziel) – unmittelbare Daten → Ziel
SUBQ # <data> von <ea>

Assembler Syntax: SUBQ.s # <data>, <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Die unmittelbaren Daten werden vom Zieloperanden subtrahiert. Der Datenbereich umfaßt 1 bis 8 (Bits 9 bis 11). Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort. Wort- und Langwortoperationen dürfen auch mit den Adreßregistern durchgeführt werden.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn der Zieloperand kleiner ist als das Datenwort (negativer Übertrag). Anderenfalls wird C gelöscht.
X wie das C-Bit.
Die Bedingungskodes werden nicht beeinflußt, wenn vom Inhalt eines Adreßregisters subtrahiert wird.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6	5 4 3 2 1 0
0	1	0	1	Daten	1	Länge	Effektive Adresse

Subtract Quick

Subtrahiere schnell

SUBQ

Befehlsfelder:

Datenfeld	Drei Bits (9 bis 11) mit direkten Daten, entsprechend einem Datenbereich von 1 bis 8 (8 ≙ 000),
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Effektives Adreßfeld	bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Veränderbar" zulässig. Hat der Operand nur Bytelänge, so ist die Adressierungsart „Adreßregister direkt" unzulässig.

SUBX

Subtract with Extend

Subtrahiere mit Erweiterungsbit

Operation:

(Ziel) – (Quelle) – X – → Ziel
SUBX Dy von Dx
SUBX Ay@– von Ax@–

Assembler Syntax:

SUBX.s Dy, Dx
SUBX.s –(Ay), –(Ax)

Operandenlänge:

s = (Byte, Wort, Langwort)

Beschreibung:

Der Quellenoperand und das Erweiterungsbit werden vom Zieloperanden subtrahiert; der Zieloperand wird mit dem Ergebnis überschrieben. Für die Adressierung der Operanden gibt es zwei Möglichkeiten:

1. Von Datenregister zu Datenregister. Die im Befehl angegebenen Datenregister enthalten die Operanden.
2. Von Speicher zu Speicher. Die Operanden werden in der Adressierungsart „Adreßregister indirekt mit Predekrement" unter Verwendung der im Befehl angegebenen Adreßregister adressiert.

Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.

Bedingungskodes:

X	N	Z	V	C
★	★	★	★	★

N wird gesetzt, wenn das Ergebnis negativ ist. Anderenfalls wird N gelöscht.
Z wird gelöscht, wenn das Ergebnis ungleich Null ist. Ist es Null, bleibt Z unverändert.
V wird gesetzt, wenn ein Überlauf entsteht. Anderenfalls wird V gelöscht.
C wird gesetzt, wenn ein negativer Übertrag entsteht. Anderenfalls wird C gelöscht.
X wie das C-Bit.

Befehlsformat:

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	0	0	1	Register Rx	1	Länge	0	0	R/M	Register Ry

Subtract with Extend

Subtrahiere mit Erweiterungsbit

SUBX

Befehlsfelder:

Registerfeld Rx	bezeichnet das Zielregister: R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")
Längenfeld	bestimmt die Länge der Operation: 00 ≙ Byte 01 ≙ Wort 10 ≙ Langwort
Bit R/M	bezeichnet die Adressierungsart des Operanden: 0 ≙ Operation erfolgt von Datenregister zu Datenregister 1 ≙ Operation erfolgt von Speicher zu Speicher
Registerfeld Ry	bezeichnet das Quellenregister R/M = 0 bedeutet Datenregister R/M = 1 bedeutet Adreßregister (für die Adressierungsart „Adreßregister indirekt mit Predekrement")

SWAP

Swap Register Halves

Vertausche Registerhälften

Operation: Registerinhalt [31—16] ↔ [15—0]
SWAP Dn

Assembler Syntax: SWAP Dn

Operandenlänge: Wort

Beschreibung:

Die beiden 16bit-Hälften eines Datenregisters werden miteinander vertauscht.

Bedingungskodes:

X	N	Z	V	C
—	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit des 32bit-Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn das 32bit-Ergebnis Null ist. Anderenfalls wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	Register		

Befehlsfelder:

Registerfeld — bestimmt das Datenregister, dessen Hälften vertauscht werden sollen.

Test and Set an Operand

Prüfe und setze Operanden

TAS

Operation: (Ziel) wird getestet → CC;
1 → Bit 7 des Ziels
TAS <ea>

Assembler Syntax: TAS <ea>

Operandenlänge: Byte

Beschreibung:

Der durch die effektive Adresse bezeichnete Operand (Bytelänge) wird getestet und gesetzt. Entsprechend diesem augenblicklichen Wert des Operanden werden N und Z gesetzt. Das höchstwertige Bit des Operanden wird gesetzt.
Dieser Befehl führt auf dem Bus des SC 68000 zu einem nicht unterbrechbaren Lese-Schreibzyklus. Er wird vorwiegend in Multiprozessorsystemen eingesetzt, wo mit gemeinsamen Datenfeldern gearbeitet wird. Das höchstwertige Bit des Operanden dient als Kennzeichnung (Semaphore) dafür, daß ein Datenfeld gerade bearbeitet wird. Der nicht unterbrechbare Buszyklus verhindert, daß zwei oder mehr Busmaster bei (quasi-)gleichzeitiger Abfrage der Semaphore von einem freigegebenen Datenfeld ausgehen (Prozessor-Synchronisation).

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn das höchstwertige Bit gesetzt ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn der Operand Null ist. Anderenfalls wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	Effektive Adresse					

Befehlsfelder:

Effektives Adreßfeld — gibt die Position des getesteten Operanden an. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

TRAP

Trap

Trap

Operation: PC → SSP@ − ; SR → SSP@ − ; (Vektor) → PC
TRAP # <Vektor>

Assembler Syntax: TRAP # <Vektor>

Operandenlänge: Keine Formatangabe

Beschreibung:

Der Prozessor beginnt mit einer Ausnahmeverarbeitung, wobei in jedem Fall mit dem SSP gearbeitet wird. Mit der Erzeugung der Vektornummer stellt er einen Bezug zum Trap-Ausnahmevektor her, der in den vier unteren Bits des Befehls angegeben ist. Sechzehn Trap-Befehlsvektoren stehen zur Verfügung.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vektor			

Befehlsfelder:

Vektorfeld — bezeichnet die Vektornummer (32—47), unter der der neue Wert des Programmzählers abgelegt ist (s. Tabelle 3).

Trap on Overflow

Trap bei Überlauf

TRAPV

Operation: Wenn V, dann TRAP
TRAPV

Assembler Syntax: TRAPV

Operandenlänge: Keine Formatangabe

Beschreibung:

Wenn ein Überlauf erfolgt ist, beginnt der Prozessor mit einer Ausnahmeverarbeitung, wobei in jedem Fall mit dem SSP gearbeitet wird. Mit der Erzeugung der Vektornummer stellt er einen Bezug zum Trap-Ausnahmevektor 7 (s. Tabelle 3) her. Ist kein Überlauf vorhanden, so wird auch keine Operation durchgeführt und der Prozessor setzt die Programmabarbeitung mit dem nächsten Befehl fort.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST

Test an Operand

Prüfe einen Operanden

Operation: (Ziel) wird getestet → CC
TST <ea>

Assembler Syntax: TST.s <ea>

Operandenlänge: s = (Byte, Wort, Langwort)

Beschreibung:

Der Operand wird mit Null verglichen, jedoch nicht verändert. Die Bedingungskodes werden entsprechend den Ergebnissen des Tests gesetzt. Die Operandenlänge beträgt entsprechend der Angabe im Längenfeld des Befehls entweder ein Byte, ein Wort oder ein Langwort.
Der Befehl TST erlaubt die gezielte Prüfung eines Operanden auf Null oder Negativ, ohne daß eine Verarbeitung oder Wertänderung stattfindet.

Bedingungskodes:

X	N	Z	V	C
–	★	★	0	0

N wird gesetzt, wenn der Operand negativ ist. Anderenfalls wird N gelöscht.
Z wird gesetzt, wenn der Operand Null ist. Ist er ungleich Null, wird Z gelöscht.
V, C } wird gelöscht.
X nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Länge		Effektive Adresse					

Befehlsfelder:

Längenfeld — bestimmt die Länge der Operation:
00 ≙ Byte
01 ≙ Wort
10 ≙ Langwort

Effektives Adreßfeld — bestimmt den Zieloperanden. Es ist nur die Adressierungskategorie „Daten ∧ Veränderbar" zulässig.

Unlink

Lösen

UNLK

Operation: An → SP; SP@+ → An
UNLK An

Assembler Syntax: UNLK An

Operandenlänge: Keine Formatangabe

Beschreibung:

Der Stack pointer wird mit dem Inhalt des angegebenen Adreßregisters geladen. Anschließend wird das Adreßregister mit dem Langwort vom oberen Ende des Stacks geladen.
Der Befehl UNLK steht in engem Zusammenhang mit dem Befehl LINK, indem er dessen Funktion umkehrt. Das Datenfeld, das zuvor mit dem Befehl LINK einem Interrupt oder einer Subroutine zugeordnet wurde, wird wieder aufgelöst, und der Stack pointer zeigt auf das Ende des nächsten Feldes. Im angegebenen Adreßregister ist nach der Operation die Anfangsadresse (−2) dieses nächsten Feldes abgelegt.
Zur Beachtung: Der Stack wird von der hohen zur niedrigen Adresse gefüllt.

Bedingungskodes:

X	N	Z	V	C
–	–	–	–	–

N, Z, V, C, X } nicht beeinflußt.

Befehlsformat:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

Befehlsfelder:

Registerfeld — bezeichnet das Adreßregister, das die Anfangsadresse (−2) des jeweiligen Speicherfeldes enthält.

7. Befehlsvorrat-Zusammenfassung

In Tabelle 15 ist der gesamte Befehlsvorrat des Mikroprozessors SC 68000 einschließlich der symbolischen Beschreibung der betreffenden Operation und der durch sie verursachten Beeinflussung der Bedingungskodes aufgelistet; Variationen von Grundbefehlen (Tabelle 5) sind eingerückt. Die Erläuterung der Abkürzungen und Definitionen findet sich in Abschnitt 6.2.

Tabelle 15. Befehlsvorrat

Mnemonic	Beschreibung	Operation	Bedingungskodes				
			X	N	Z	V	C
ABCD	Addiere dezimal mit Erweiterungsbit	$(\text{Ziel})_{10} + (\text{Quelle})_{10} + X \rightarrow \text{Ziel}$	★	U	★	U	★
ADD	Addiere binär	(Ziel) + (Quelle) → Ziel	★	★	★	★	★
ADDA	Addiere Adresse	(Ziel) + (Quelle) → Ziel	–	–	–	–	–
ADDI	Addiere unmittelbar	(Ziel) + unmittelbare Daten → Ziel	★	★	★	★	★
ADDQ	Addiere schnell	(Ziel) + unmittelbare Daten → Ziel	★	★	★	★	★
ADDX	Addiere mit Erweiterungsbit	(Ziel) + (Quelle) + X → Ziel	★	★	★	★	★
AND	Logisches UND	(Ziel) ∧ (Quelle) → Ziel	–	★	★	0	0
ANDI	Logisches UND unmittelbar	(Ziel) ∧ unmittelbare Daten → Ziel	–	★	★	0	0
ASL, ASR	Arithmetische Verschiebung	(Ziel) verschoben um <Stellenzahl> → Ziel	★	★	★	★	★
B_{CC}	Verzweige bedingt	Wenn $_{CC}$, dann PC + d → PC	–	–	–	–	–
BCHG	Prüfe ein Bit und ändere es	~ (<Bitnummer> vom Ziel) → Z ~ (<Bitnummer> vom Ziel) → <Bitnummer> vom Ziel	–	–	★	–	–

Fortsetzung Tabelle 15. Befehlsvorrat

Mnemonic	Beschreibung	Operation	Bedingungs-kodes X	N	Z	V	C
BCLR	Prüfe ein Bit und lösche es	~ (<Bitnummer> vom Ziel) → Z 0 → <Bitnummer> → vom Ziel	–	–	★	–	–
BRA	Verzweige immer	PC + d → PC	–	–	–	–	–
BSET	Prüfe ein Bit und setze es	~ (<Bitnummer> vom Ziel) → Z 1 → <Bitnummer> vom Ziel	–	–	★	–	–
BSR	Verzweige zur Subroutine	PC → SP@ – ; PC + d → PC	–	–	–	–	–
BTST	Prüfe ein Bit	~ (<Bitnummer> vom Ziel) → Z	–	–	★	–	–
CHK	Prüfe Register auf Grenzen	Wenn Dn <0 oder Dn> (<ea>), dann TRAP	–	★	U	U	U
CLR	Lösche einen Operanden	0 → Ziel	–	0	1	0	0
CMP	Vergleiche	(Ziel) – (Quelle)	–	★	★	★	★
CMPA	Vergleiche Adresse	(Ziel) – (Quelle)	–	★	★	★	★
CMPI	Vergleiche unmittelbar	(Ziel) – unmittelbare Daten	–	★	★	★	★
CMPM	Vergleiche Speicher	(Ziel) – (Quelle)	–	★	★	★	★
DB_{CC}	Prüfe Bedingung, dekrementiere und verzweige	Wenn CC, dann Dn – 1 → Dn; wenn Dn ≠ –1, dann PC + d → PC	–	–	–	–	–
DIVS	Division mit Vorzeichen	(Ziel)/(Quelle) → Ziel	–	★	★	★	0
DIVU	Division ohne Vorzeichen	(Ziel)/(Quelle) → Ziel	–	★	★	★	0
EOR	Logisches exklusives ODER	(Ziel) ⊻ (Quelle) → Ziel	–	★	★	0	0

Fortsetzung Tabelle 15. Befehlsvorrat

Mnemonic	Beschreibung	Operation	Bedingungs-kodes				
			X	N	Z	V	C
EORI	Exklusives ODER unmittelbar	(Ziel) ≡ unmittelbare Daten → Ziel	–	★	★	0	0
EXG	Vertausche Daten zwischen Registern	Rx ↔ Ry	–	–	–	–	–
EXT	Vorzeichenerweiterung	(Ziel) vorzeichen-erweitert → Ziel	–	★	★	0	0
ILLEGAL	Unzulässiger Befehl (TRAP)	PC → SSP@ – ; SR → SSP@ – ; (Vektor) → PC	–	–	–	–	–
JMP	Springe	Ziel → PC	–	–	–	–	–
JSR	Springe zur Subroutine	PC → SP@ – ; Ziel → PC	–	–	–	–	–
LEA	Lade die effektive Adresse	Ziel → An	–	–	–	–	–
LINK	Anbinden und Reservieren	An → SP@ – ; SP → An; SP + d → SP	–	–	–	–	–
LSL, LSR	Logische Verschiebung	(Ziel) verschoben um <Stellenzahl> → Ziel	★	★	★	0	★
MOVE	Transportiere Daten von der Quelle zum Ziel	(Quelle) → Ziel	–	★	★	0	0
MOVE to CCR	Transportiere zum Bedingungskoderegister	(Quelle) → CCR	★	★	★	★	★
MOVE to SR	Transportiere zum Status-register	(Quelle) → SR	★	★	★	★	★
MOVE from SR	Transportiere vom Status-register	SR → Ziel	–	–	–	–	–
MOVE USP	Transportiere den Inhalt des Anwender-Stack pointers	USP → An; An → USP	–	–	–	–	–
MOVEA	Transportiere Adresse	(Quelle) → Ziel	–	–	–	–	–
MOVEQ	Transportiere schnell	unmittelbare Daten → Ziel	–	★	★	0	0

Fortsetzung Tabelle 15. Befehlsvorrat

Mnemonic	Beschreibung	Operation	Bedingungs-kodes X	N	Z	V	C
MOVEM	Transportiere mehrere Register	Mehrere Register → Ziel (Quelle) → mehrere Register	–	–	–	–	–
MOVEP	Transportiere periphere Daten	(Quelle) → Ziel	–	–	–	–	–
MULS	Multiplikation mit Vorzeichen	(Ziel) × (Quelle) → Ziel	–	★	★	0	0
MULU	Multiplikation ohne Vorzeichen	(Ziel) × (Quelle) → Ziel	–	★	★	0	0
NBCD	Negiere dezimal mit Erweiterungsbit	$0 - (\text{Ziel})_{10} - X \rightarrow$ Ziel	★	U	★	U	★
NEG	Negiere	0 – (Ziel) → Ziel	★	★	★	★	★
NEGX	Negiere mit Erweiterungsbit	0 – (Ziel) – X → Ziel	★	★	★	★	★
NOP	Keine Operation	–	–	–	–	–	–
NOT	Einer-Komplement	~ (Ziel) → Ziel	–	★	★	0	0
OR	Logisches ODER	(Ziel) v (Quelle) → Ziel	–	★	★	0	0
ORI	Logisches ODER unmittelbar	(Ziel) v unmittelbare Daten → Ziel	–	★	★	0	0
PEA	Ablegen der effektiven Adresse	Ziel → SP@ –	–	–	–	–	–
RESET	Setze externe Einheiten zurück	–	–	–	–	–	–
ROL, ROR	Rundverschiebung ohne Erweiterungsbit	(Ziel) rundgeschoben um <Stellenzahl> → Ziel	–	★	★	0	★
ROXL, ROXR	Rundverschiebung mit Erweiterungsbit	(Ziel) rundgeschoben um <Stellenzahl> → Ziel	★	★	★	0	★
RTE	Springe von Ausnahme zurück	SP@ + → SR; SP@ + → PC	★	★	★	★	★

Fortsetzung Tabelle 15. Befehlsvorrat

Mnemonic	Beschreibung	Operation	Bedingungs-kodes X	N	Z	V	C
RTR	Springe zurück und lade Bedingungskodes	SP@ + → CC; SP@ + → PC	★	★	★	★	★
RTS	Springe zurück von der Subroutine	SP@ + → PC	–	–	–	–	–
SBCD	Subtrahiere dezimal mit Erweiterungsbit	$(\text{Ziel})_{10} - (\text{Quelle})_{10} - X$ → Ziel	★	U	★	U	★
S_{CC}	Setze Byte bedingt	Wenn $_{CC}$, dann Einsen → Ziel; sonst Nullen → Ziel	–	–	–	–	–
STOP	Lade das Statusregister und stoppe	unmittelbare Daten → SR; STOP	★	★	★	★	★
SUB	Subtrahiere binär	(Ziel) – (Quelle) → Ziel	★	★	★	★	★
SUBA	Subtrahiere Adresse	(Ziel) – (Quelle) → Ziel	–	–	–	–	–
SUBI	Subtrahiere unmittelbar	(Ziel) – unmittelbare Daten → Ziel	★	★	★	★	★
SUBQ	Subtrahiere schnell	(Ziel) – unmittelbare Daten → Ziel	★	★	★	★	★
SUBX	Subtrahiere mit Erweiterungsbit	(Ziel) – (Quelle) – X → Ziel	★	★	★	★	★
SWAP	Vertausche Registerhälften	Registerinhalt [31—16] ↔ [15—0]	–	★	★	0	0
TAS	Prüfe und setze Operanden	(Ziel) wird getestet → CC 1 → [7] des Ziels	–	★	★	0	0
TRAP	Trap	PC → SSP@ – ; SR → SSP@ – ; (Vektor) → PC	–	–	–	–	–
TRAPV	Trap bei Überlauf	Wenn V, dann TRAP	–	–	–	–	–
TST	Prüfe einen Operanden	(Ziel) wird getestet → CC	–	★	★	0	0
UNLK	Lösen	An → SP; SP@ + → An	–	–	–	–	–

Erklärung zu den Operationen (Tabelle 15)

An	Adreßregister	SR	Statusregister
@	indirekt	SSP	Supervisor-Stack pointer
@+	indirekt mit Postinkrement	USP	Anwender-Stack pointer
@−	indirekt mit Predekrement	X	Erweiterung
CC	Bedingungskode	[ ]	Bitnummern
CCR	Bedingungskoderegister (Anwender-Byte)	⋕	exklusives ODER (Antivalenz)
		( )	Inhalt von
Dn	Datenregister	~	Invertierung
d	Adreßdistanz	∧	logisch UND
SP	Stack pointer	v	logisch ODER
		×	Multiplikation

Erklärung zu den Bedingungskodes (Tabelle 15)

X	Erweiterung	★	Bit wird beeinflußt
N	Negativ	–	Bit wird nicht beeinflußt
Z	Null	0	Bit wird 0
V	Überlauf	1	Bit wird 1
C	Übertrag	U	nicht definiert

8. Berechnung der Bedingungskodes

8.1. Allgemeines

Dieser Abschnitt beschreibt, wie die Bedingungskodes beeinflußt werden, was die einzelnen Bits bedeuten, wie sie berechnet werden und wie sie bei der detaillierten Befehlsbeschreibung (Abschnitt 6.3) dargestellt sind.

Für die Beeinflussung der Bedingungskodes sind zwei Kriterien maßgebend:

- Übereinstimmung — bezüglich Befehlsvarianten, Befehlsanwendungen und bedingter Befehle;
- signifikante Ergebnisse — keine Änderungen, sofern nicht nützliche Informationen geliefert werden.

Übereinstimmung bezüglich Befehlsvarianten bedeutet, daß Varianten von Grundbefehlen die Bedingungskodes auf dieselbe Weise beeinflussen. Übereinstimmung bezüglich Befehlsanwendungen bedeutet, daß die Beeinflussung der Bedingungskodes durch einen bestimmten Befehl bei jeder Anwendung dieses Befehls in gleicher Weise erfolgt. Übereinstimmung bezüglich bedingter Befehle bedeutet, daß unabhängig davon, ob die Bedingungskodes durch einen Vergleichs-, Test- oder Transportbefehl gesetzt wurden, bedingte Befehle dieselbe Situation vorfinden. Die möglichen Bedingungen für die bedingten Befehle (B_{CC}, DB_{CC} und S_{CC}) sind in Tabelle 13 aufgelistet.

8.2. Bedingungskoderegister

Das Bedingungskoderegister besteht aus den fünf unteren Bits des Statusregisters, wie in Bild 5 dargestellt ist. In Abschnitt 6.3 (Befehlsbeschreibung) ist es stets in folgender Form angegeben:

X	N	Z	V	C

Die einzelnen Bits bedeuten

N (Negativ)	wird gesetzt, wenn das höchstwertige Bit eines Ergebnisses gesetzt ist. Anderenfalls wird N gelöscht (Ausnahmen s. Tabelle 16).
Z (Null)	wird gesetzt, wenn das Ergebnis Null ist. Ist es ungleich Null, wird Z gelöscht (Ausnahmen s. Tabelle 16).
V (Überlauf)	wird gesetzt, wenn ein arithmetischer Überlauf entsteht. (Übergang von FFFF nach 8000.) Dies bedeutet, daß das Ergebnis im vorgegebenen Operandenformat nicht dargestellt werden kann. Anderenfalls wird V gelöscht.
C (Übertrag)	wird gesetzt, wenn bei einer Addition ein Übertrag aus den höchstwertigen Bits der Operanden erzeugt wird. Dasselbe gilt, wenn bei einer Subtraktion ein negativer Übertrag erzeugt wird. Anderenfalls wird C gelöscht (Einzelheiten s. Tabelle 16).
X (Erweiterung)	Unbeeinflußt bei Datentransport. Wenn es beeinflußt wird, geschieht dies in gleicher Weise wie beim C-Bit (Einzelheiten s. Tabelle 16).

Die ersten vier Bits des Bedingungskoderegisters (C, V, Z, N) geben die eigentlichen Bedingungen wieder, die aus der Operation des Prozessors folgen. Das X-Bit ist ein Erweiterungsoperand für Rechnungen mit mehrfacher Genauigkeit und erspart beim Ablegen oder Holen von Teilergebnissen innerhalb einer komplexen Operation ein zwischenzeitliches Retten des Übertragsbits C.

Die Bedingungskodes sind in Abschnitt 6.3 wie folgt gekennzeichnet:

★ Bit wird beeinflußt,
– Bit wird nicht beeinflußt,
0 Bit wird 0,
1 Bit wird 1,
U nicht definiert.

8.3. Tabelle zur Berechnung der Bedingungskodes

Die meisten Operationen verwenden einen Quellen- und einen Zieloperanden, rechnen und legen das Ergebnis im Ziel ab. Bei Einoperanden-Operationen wird nur der Zieloperand verwendet und berechnet; das Ergebnis wird ebenfalls im Ziel abgelegt. In Tabelle 16 ist angegeben, in welcher Weise die Bedingungskodes durch die einzelnen Befehle gesetzt werden.

Tabelle 16. Berechnung der Bedingungskodes

Operationen	X	N	Z	V	C	Spezielle Definition
ABCD	★	U	?	U	?	C = Dezimaler Übertrag $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
ADD, ADDI, ADDQ	★	★	★	?	?	$V = (Sm \wedge Dm \wedge \overline{Rm}) \vee (\overline{Sm} \wedge \overline{Dm} \wedge Rm)$ $C = (Sm \wedge Dm) \vee (\overline{Rm} \wedge Dm) \vee (Sm \wedge \overline{Rm})$
ADDX	★	★	?	?	?	$V = (Sm \wedge Dm \wedge \overline{Rm}) \vee (\overline{Sm} \wedge \overline{Dm} \wedge Rm)$ $C = (Sm \wedge Dm) \vee (Rm \wedge Dm) \vee (Sm \wedge \overline{Rm})$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	–	★	★	0	0	
CHK	–	★	U	U	U	
SUB, SUBI SUBQ	★	★	★	?	?	$V = (\overline{Sm} \wedge Dm \wedge \overline{Rm}) \vee (Sm \wedge \overline{Dm} \wedge Rm)$ $C = (Sm \wedge \overline{Dm}) \vee (Rm \wedge \overline{Dm}) \vee (Sm \wedge Rm)$
SUBX	★	★	?	?	?	$V = (\overline{Sm} \wedge Dm \wedge \overline{Rm}) \vee (Sm \wedge \overline{Dm} \wedge Rm)$ $C = (Sm \wedge \overline{Dm}) \vee (Rm \wedge \overline{Dm}) \vee (Sm \wedge Rm)$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge R0$
CMP, CMPI, CMPM	–	★	★	?	?	$V = (\overline{Sm} \wedge Dm \wedge \overline{Rm}) \vee (Sm \wedge \overline{Dm} \wedge Rm)$ $C = (Sm \wedge \overline{Dm}) \vee (Rm \wedge \overline{Dm}) \vee (Sm \wedge Rm)$
DIVS, DIVU	–	★	★	?	0	V = Überlauf bei Division
MULS, MULU	–	★	★	0	0	
SBCD, NBCD	★	U	?	U	?	C = Negativer dezimaler Übertrag $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
NEG	★	★	★	?	?	$V = Dm \wedge Rm$, $C = Dm \vee Rm$
NEGX	★	★	?	?	?	$V = Dm \wedge Rm$, $C = Dm \vee Rm$ $Z = Z \wedge \overline{Rm} \wedge \ldots \wedge \overline{R0}$
BTST, BCHG, BSET, BCLR	–	–	?	–	–	$Z = \overline{Dn}$

Fortsetzung Tabelle 16. Berechnung der Bedingungskodes

Operationen	X	N	Z	V	C	Spezielle Definition
ASL	★	★	★	?	?	$V = Dm \wedge (\overline{D_{m-1}} \vee \ldots \vee \overline{D_{m-r}}) \vee \overline{Dm} \wedge (D_{m-1} \vee \ldots \vee D_{m-r})$ $C = D_{m-r+1}$
ASL (r = 0)	–	★	★	0	0	
LSL, ROXL	★	★	★	0	?	$C = D_{m-r+1}$
LSR (r = 0)	–	★	★	0	0	
ROXL (r = 0)	–	★	★	0	?	$C = X$
ROL	–	★	★	0	?	$C = D_{m-r+1}$
ROL (r = 0)	–	★	★	0	0	
ASR, LSR, ROXR	★	★	★	0	?	$C = D_{r-1}$
ASR, LSR (r = 0)	–	★	★	0	0	
ROXR (r = 0)	–	★	★	0	?	$C = X$
ROR	–	★	★	0	?	$C = D_{r-1}$
ROR (r = 0)	–	★	★	0	0	

– nicht beeinflußt,
U nicht definiert,
? s. spezielle Definition,
★ allgemeiner Fall:
$X = C$
$N = Rm$
$Z = \overline{Rm} \wedge \ldots \wedge \overline{R0}$

Sm höchstwertiges Bit im Quellenoperanden,
Dm höchstwertiges Bit im Zieloperanden,
Rm höchstwertiges Bit des Ergebnisses,
r Stellenzahl, um die geschoben wird.

9. Mikroprogramm-Architektur des SC 68000

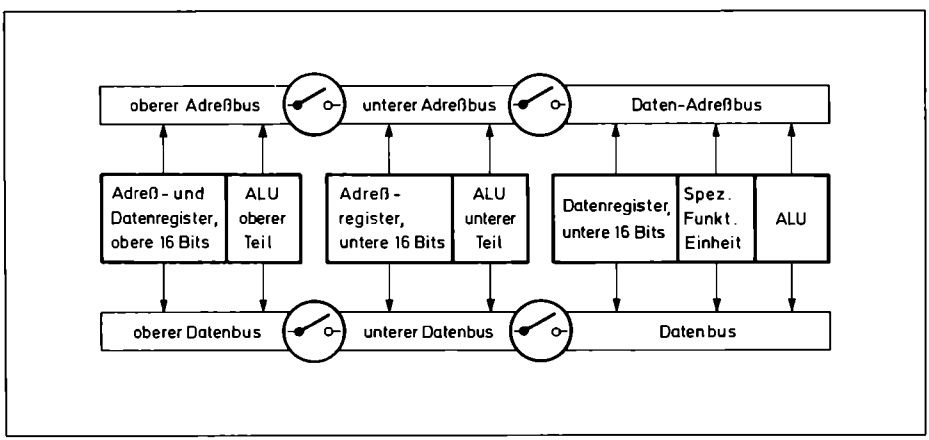
Erstmals wurde bei den Mikroprozessoren der Familie 68000 die aus dem Bereich der Groß- und Minicomputer bekannte Mikroprogrammierung realisiert. Gegenüber den herkömmlichen Entwicklungen mit Standard-Logikfunktionen bietet die Mikroprogrammierung erhebliche Vorteile, wobei es die moderne VLSI-Technik ermöglicht hat, einen Mikroprogramm-Speicher genügender Größe auf dem Kristall zu integrieren. Die Vorteile sind:

- reguläre Layout-Strukturen (= hohe Packungsdichte),
- klare Zeitverhältnisse im Funktionsablauf (= gute Überschaubarkeit des Designs),
- parallele Entwicklung der Hardware- und Befehls-Architektur (= Verkürzung der Entwicklungszeit),
- leichte Änderbarkeit (= schnelle Korrektur von Designfehlern),
- einfache Erweiterbarkeit (= Erfüllen von Forderungen aus dem praktischen Einsatz),
- Befehlsvorrat-Optimierung (= vereinfachte Entwicklung von Spezialprozessoren).

Die Registerarchitektur des SC 68000 ist in drei Bereiche aufgeteilt, die alle über interne Adreß- und Datenbusse miteinander kommunizieren (Bild 17).

Der SC 68000 benutzt eine vertikale Mikroprogramm-Struktur. Der Mikroprogramm-Speicher enthält eine Sequenz von Adressen, die jedem Befehl des Mikroprozessors zugeordnet sind. Derzeit sind etwa 300 der 650 Adressen ausgenutzt. Die eigentliche Steuerung der Ausführungseinheit wird über einen vom Mikroprogramm adressierten Nanoprogramm-Speicher durchgeführt, der einen nach sinnvollen Abläufen aufgebauten Satz von nicht duplizierten Steuerwörtern enthält (Bild 18).

Bild 17. Interne Ausführungseinheit des SC 68000



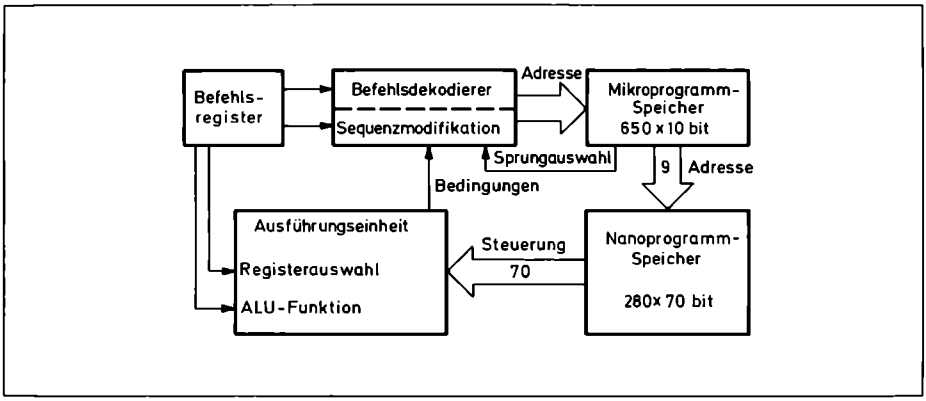


Bild 18. Mikroprogramm-Struktur des SC 68000

Einige der notwendigen Informationen für die Ausführungseinheit werden direkt dem Befehlskode entnommen und damit nicht in den Mikroprogramm-Ablauf einbezogen — so z. B. die Auswahl eines Registers oder interne ALU-Funktionen.

10. Prefetch-Mechanismus

10.1. Allgemeines

Zur weiteren Verbesserung der Leistungsfähigkeit besitzt der SC 68000 einen Vorhol(Prefetch)-Mechanismus. Dieser Mechanismus läßt sich durch die Operationen der jeweils ablaufenden Mikroprogramme beschreiben:

Wenn die Ausführung eines Befehls beginnt, sind das Operationswort (Befehlskode) und das folgende Wort bereits aus dem Programmspeicher gelesen worden. Das Operationswort befindet sich im Befehlsdekodierer.

Wenn es sich bei dem Befehl um einen Einwort-Sprungbefehl handelt (z. B. BRA mit 8bit-Adreßdistanz), wird das nächste (bereits empfangene) Wort nicht benutzt. Im Falle eines externen Interrupts oder der Ausnahme TRACE werden beide Wörter nicht benutzt. Der Programmzähler (PC) zeigt normalerweise auf die zuletzt adressierte Speicherstelle und wird erst unmittelbar vor dem nächsten Adressierungsvorgang auf den aktuellen Wert gebracht.

10.2. Befehls-Prefetch

Das folgende Beispiel zeigt die Wirkungsweise des Befehls-Prefetch, wobei von einem willkürlichen Programm ausgegangen wird:

```
        ORG     0
        DC.L    INITSSP     ; Wert für Supervisor-Stack pointer
        DC.L    START       ; Wert für Programmzähler

        ORG     INTVEKTOR   ; Definiere Interrupt-Vektor
        DC.L    INTPROG     ; Interrupt-Startadresse

        ORG     START       ; Programmanfang

START:  NOP                 ; Keine Operation
        BRA.S   LABEL       ; Kurzer Sprung
        ADD.W   D0, D1      ; Addiere zwei Registerinhalte
LABEL:  SUB.W   DIST(A0), A1 ; Subtrahiere Speicherinhalt von
                            ; Registerinhalt
        CMP.W   D2, D3      ; Vergleiche zwei Registerinhalte
        SGE.B   D7          ; Setze, wenn größer – gleich
        -----
        -----

INTPROG: MOVE.W ADR1, ADR2  ; Transportiere Speicher zu Speicher
        NOP                 ; Keine Operation
        SWAP.W  D0          ; Vertausche Wörter
        -----
        -----
```

Das vorstehende Programm beginnt nach dem Einschalten des SC 68000, wobei angenommen wird, daß während der Ausführung des Befehls CMP.W ein Interrupt eintritt.

Mikroprogramm-Ablauf	Operation	Adresse	Operand
Reset-Ende			
	Lesen	0	SSP oberer Teil
	Lesen	2	SSP unterer Teil
	Lesen	4	PC oberer Teil
	Lesen	6	PC unterer Teil
	Lesen	(PC)	NOP
	Lesen	+(PC)	BRA
Beginn NOP	Lesen	+(PC)	ADD
Beginn BRA	PC = PC + d		
	Lesen	(PC)	SUB
	Lesen	+(PC)	DIST
Beginn SUB	Lesen	+(PC)	CMP
	Lesen	DIST(A0)	(Speicherstelle)
	Lesen	+(PC)	SGE
Beginn CMP	Interrupt		
Interrupt	Schreiben	−(SSP)	PC unterer Teil
	Schreiben	−(SSP)	PC oberer Teil
	Lesen		Vektornummer
	Schreiben	−(SSP)	Statusregister
	Lesen	(VR)	PC oberer Teil
	Lesen	+(VR)	PC unterer Teil
	Lesen	(PC)	MOVE
	Lesen	+(PC)	ADR1 oberer Teil
Beginn MOVE	Lesen	+(PC)	ADR1 unterer Teil
	Lesen	+(PC)	ADR2 oberer Teil
	Lesen	ADR1	(ADR1)
	Lesen	+(PC)	ADR2 unterer Teil
	Lesen	+(PC)	NOP
	Schreiben	ADR2	(ADR1)
	Lesen	+(PC)	SWAP
Beginn NOP			

Erklärung der Abkürzungen

( ) ≙ Inhalt von
VR ≙ Vektorregister
+ ≙ Preinkrement
− ≙ Predekrement

10.3. Daten-Prefetch

Normalerweise ist der Prefetch-Mechanismus des SC 68000 nur bei Befehlen und nicht bei Daten wirksam. Eine Ausnahme bildet der Befehl MOVEM (Transportiere mehrere Register), bei dem auch Daten in den Prefetch einbezogen werden, wenn der Speicher Datenquelle ist:

```
MOVEM.L     ADR, D0/D1  ; Lies vier Wörter ab
                        ; Adresse ADR und
                        ; lege diese in den
                        ; Registern D0 und D1 ab
```

Unter der Annahme, daß die Berechnung der effektiven Adresse bereits durchgeführt ist, laufen intern folgende Operationen ab:

Mikroprogramm Ablauf	Operation extern	Adresse	Operation intern
Beginn MOVEM	Lesen	ADR	
	Lesen	ADR+2	(ADR) → D0 oberer Teil
	Lesen	ADR+4	(ADR+2) → D0 unterer Teil
	Lesen	ADR+6	(ADR+4) → D1 oberer Teil
	Lesen	ADR+8	(ADR+6) → D1 unterer Teil

Der Inhalt von Speicherstelle ADR+8 wurde zwar gelesen, wird aber nicht mehr verarbeitet.

11. Programmbeispiele

Programmbeispiel 1

```
                    !
                    ! 32 * 32 BIT MULTIPLIKATION OHNE VORZEICHEN
                    !
                    ! DIE BEIDEN 32 BIT MULTIPLIKANTEN STEHEN IN
                    ! D0 UND D1
                    ! DAS 64 BIT ERGEBNIS STEHT IN D4 UND D5 (D4 = MSB)
                    !
                    !
                    ! RECHENVERFAHREN:
                    !
                    !((D0.31-16) (D0.15-0)) * ((D1.31-16) (D1.15-0)) =
                    !
                    ! D0.15- 0 * D1.15- 0 =                (E1.1  E1.2)
                    ! D0.31-16 * D1.15- 0 =          (E2.1  E2.2)
                    ! D0.15- 0 * D1.31-16 =          (E3.1  E3.2)
                    ! D0.31-16 * D1.31-16 =   (E4.1  E4.2)
                    !                          ------------------------
                    !                          ( D4.31- 0 )( D5.31- 0 )
                    !
                    ORG $1050;          ! PROGRAMMSTARTADRESSE
                    !
1050    2400            MOVE.L D0,D2; ! D0 ZWISCHENSPEICHERN
1052    2600            MOVE.L D0,D3; ! D0 ZWISCHENSPEICHERN
1054    2801            MOVE.L D1,D4; ! D1 ZWISCHENSPEICHERN
1056    4843            SWAP   D3;    ! D3 = (D0.15- 0 D0.31-16)
1058    4844            SWAP   D4;    ! D4 = (D1.15- 0 D1.31-16)
105A    C0C1            MULU   D1,D0; ! D0 = (E1.1  E1.2)
105C    C2C3            MULU   D3,D1; ! D1 = (E2.1  E2.2)
105E    C4C4            MULU   D4,D2; ! D2 = (E3.1  E3.2)
1060    C6C4            MULU   D4,D3; ! D3 = (E4.1  E4.2)
1062    4840            SWAP   D0;    ! D0 = (E1.2  E1.1)
1064    D041            ADD.W  D1,D0; ! D0 =  E1.1 + E2.2
1066    4284            CLR.L  D4;
1068    D784            ADDX.L D4,D3; ! (E4.1 E4.2) = (E4.1 E4.2)
                                      !+ UEBERTRAG AUS D0 + D1
106A    D042            ADD.W  D2,D0; ! D0 = (E1.1 +E2.2) + E3.2
106C    D784            ADDX.L D4,D3; ! (E4.1 E4.1) = (E4.1 E4.2)
                                      !+ UEBERTRAG AUS D0 + D2
106E    4840            SWAP   D0;    ! DIE 32 NIEDERWERTIGSTEN BITS
                                      !DES ERGEBNISSES STEHEN IN D0
1070    2A00            MOVE.L D0,D5; ! ERGEBNIS IN D5 ABSPEICHERN
1072    4241            CLR.W  D1;    ! E2.2 = 0
1074    4242            CLR.W  D2;    ! E3.2 = 0
1076    4841            SWAP   D1;    ! D1 = (E2.2  E2.1)
1078    4842            SWAP   D2;    ! D2 = (E3.2  E3.1)
107A    D282            ADD.L  D2,D1; ! D1 = E2.1 +E3.1
107C    D283            ADD.L  D3,D1; ! D1 = (E2.1 + E3.1) + E4.1-2
                                      !DIE 32 HOECHSTWERTIGSTEN BITS
                                      !DES ERGEBNISSES STEHEN IN D1
107E    2801            MOVE.L D1,D4; ! ERGEBNIS IN D4 ABSPEICHERN
1080    4E40            TRAP   0;     ! RUECKSPRUNG ZUM MONITOR
```

Programmbeispiel 2

```
                    !
                    ! 32 * 32 BIT MULTIPLIKATION MIT VORZEICHEN
                    !
                    ! DIE BEIDEN 32 BIT MULTIPLIKANTEN STEHEN IN
                    ! D0 UND D1
                    ! DAS 64 BIT ERGEBNIS STEHT IN D6 UND D7 (D6 = MSB)
                    !
                    !
```

Fortsetzung Programmbeispiel 2

```
                                  ! RECHENVERFAHREN:
                                  !
                                  ! ((D0.31-16) (D0.15-0)) * ((D1.31-16) (D1.15-0)) =
                                  !
                                  ! D0.15- 0 * D1.15- 0 =                (E1.1  E1.2)
                                  ! D0.31-16 * D1.15- 0 =          (E2.1  E2.2)
                                  ! D0.15- 0 * D1.31-16 =          (E3.1  E3.2)
                                  ! D0.31-16 * D1.31-16 =   (E4.1  E4.2)
                                  !                          ------------------------
                                  !                          ( D6.31- 0 )( D7.31- 0 )
                                  ! WENN D0 NEGATIV DANN   -( D1.31- 0 )
                                  ! WENN D1 NEGATIV DANN   -( D0.31- 0 )
                                  !                          ------------------------
                                  !                          ( D6.31- 0 )( D7.31- 0 )
                                  !
                                  !
                                  ORG $10A0;              ! PROGRAMMSTARTADRESSE
                                  !
  10A0    2400                          MOVE.L D0,D2;  ! D0 ZWISCHENSPEICHERN
  10A2    2600                          MOVE.L D0,D3;  ! D0 ZWISCHENSPEICHERN
  10A4    2801                          MOVE.L D1,D4;  ! D1 ZWISCHENSPEICHERN
  10A6    2A00                          MOVE.L D0,D5;  ! D0 ZWISCHENSPEICHERN
  10A8    4843                          SWAP   D3;     ! D3 = (D0.15- 0 D0.31-16)
  10AA    4844                          SWAP   D4;     ! D4 = (D1.15- 0 D1.31-16)
  10AC    C0C1                          MULU   D1,D0;  ! D0 = (E1.1  E1.2)
  10AE    C2C3                          MULU   D3,D1;  ! D1 = (E2.1  E2.2)
  10B0    C4C4                          MULU   D4,D2;  ! D2 = (E3.1  E3.2)
  10B2    C6C4                          MULU   D4,D3;  ! D3 = (E4.1  E4.2)
  10B4    4840                          SWAP   D0;     ! D0 = (E1.2  E1.1)
  10B6    D041                          ADD.W  D1,D0;  ! D0 = E1.1 + E2.2
  10B8    4287                          CLR.L  D7;
  10BA    D787                          ADDX.L D7,D3;  ! (E4.1 F4.2) = (E4.1 E4.2)
                                                       !+ UEBER.RAG AUS D0 + D1
  10BC    D042                          ADD.W  D2,D0;  ! D0 = (E1.1 + E2.2) + E3.2
  10BE    D787                          ADDX.L D7,D3;  ! (E4.1 E4.2) = (E4.1 E4.2)
                                                       !+ UEBERTRAG AUS D0 + D2
  10C0    4840                          SWAP   D0;     ! DIE 32 NIEDERWERTIGSTEN
                                                       !BITS DES ERGEBNISSES STEHEN
                                                       !IN D0
  10C2    2E00                          MOVE.L D0,D7;  ! ERGEBNIS IN D7 ABSPEICHERN
  10C4    4241                          CLR.W  D1;     ! E2.2 = 0
  10C6    4242                          CLR.W  D2;     ! E3.2 = 0
  10C8    4841                          SWAP   D1;     ! D1 = (E2.2  E2.1)
  10CA    4842                          SWAP   D2;     ! D2 = (E3.2  E3.1)
  10CC    D282                          ADD.L  D2,D1;  ! D1 = E2.1 + E3.1
  10CE    D283                          ADD.L  D3,D1;  ! D1 = (E2.1 + E3.1) + E4.1-2
  10D0    2C01                          MOVE.L D1,D6;  ! DIE 32 HOECHSTWERTIGSTEN
                                                       !BITS DES ZWISCHENERGEBNISSES
                                                       !IN D6 ABSPEICHERN
  10D2    4844                          SWAP   D4;     ! IN D4 STEHT DER ZWEITE MUL-
                                                       !TIPLIKANT
  10D4    4A85                          TST.L  D5;     ! TESTEN OB DER ERSTE MULTI-
  10D6    6A02                          BPL.S  MULTI2;!PLIKANT POSITIV IST
  10D8    9C84                          SUB.L  D4,D6;  ! D6 = D6 - ZWEITEN MULTIPLI-
                                                       !KANTEN
  10DA    4A84                  MULTI2: TST.L  D4;     ! TESTEN OB DER ZWEITE MULTI-
  10DC    6A02                          BPL.S  FERTIG;!PLIKANT POSITIV IST
  10DE    9C85                          SUB.L  D5,D6;  ! D6 = D6 - ERSTEN MULTIPLI-
                                                       !KANTEN
  10E0    4E40                  FERTIG: TRAP   0;;     ! RUECKSPRUNG ZUM MONITOR
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT	
FERTIG		69		$10E0	PC-R.
MULTI2		65		$10DA	PC-R.

Programmbeispiel 3

```
                    DIVISION OHNE VORZEICHEN MIT
                    UEBERLAUF (QUOTIENT GROESSER FFFF)

                    DER DIVIDEND STEHT IN D4
                    DER DIVISOR STEHT IN D5
                    DER QUOTIENT DES ERGEBNISSES STEHT IN D6
                    DER REST DES ERGEBNISSES STEHT IN D7

                    RECHENVERFAHREN:

                    B = BIT   R = REST   Q = QUOTIENT   V = 1 = UEBERLAUF

                  !((B 31-16)(B 15-0)) / (B 15- 0)=((B 31-16)(B 15-0))
                  !-----------------------------------------------------

                  !(      DIVIDEND     ) / (DIVISOR)=(( R 1  )( Q 1 ))


                  ! WENN V = 1 DANN

                  !((   0   )(  R 1 )) / (DIVISOR)=((  R 2  )(  Q 2 ))
                  !((  R 2  )(  Q 1 )) / (DIVISOR)=((  R 3  )(  Q 3 ))

                  ! ERGEBNIS  =  (         QUOTIENT         )(   REST  )
                  !              ((    Q 2     )(    Q 3    ))(   R 3   )
                  !              (( D6.31-16 )( D6.15- 0 ))( D7.15-0 )


                    WENN V = 0 DANN

                  ! ERGEBNIS  =  (         QUOTIENT         )(   REST  )
                  !              ((     0      )(    Q 1    ))(   R 1   )
                  !              (( D6.31-16 )( D6.15- 0 ))( D7.15-0 )


                   ORG $1200;

  1200    2004                 MOVE.L D4,D0;     ! DIVIDEND RETTEN
  1202    2205                 MOVE.L D5,D1;     ! DIVISOR RETTEN
  1204    4283                 CLR.L  D3;
                                                 ! DX = (B 31-16)(B 15- 0)
                                                 !-----------------------
  1206    80C1                 DIVU   D1,D0;     ! D0 = (  R 1  )(  Q 1  )
  1208    680E                 BVC.S  V_NULL;    !  PRUEFEN OB V = 1
  120A    2400                 MOVE.L D0,D2;     ! D2 = (  R 1  )(  Q 1  )
  120C    4240                 CLR.W  D0;        ! D0 = (  R 1  )(   0   )
  120E    4840                 SWAP   D0;        ! D0 = (   0   )(  R 1  )
  1210    80C1                 DIVU   D1,D0;     ! D0 = (  R 2  )(  Q 2  )
  1212    3600                 MOVE.W D0,D3;     ! D3 = (   0   )(  Q 2  )
  1214    3002                 MOVE.W D2,D0;     ! D0 = (  R 2  )(  Q 1  )
  1216    80C1                 DIVU   D1,D0;     ! D0 = (  R 3  )(  Q 3  )
  1218    2200        V_NULL:  MOVE.L D0,D1;     ! D1 = (  R 3  )(  Q 3  )
  121A    4840                 SWAP   D0;        ! D0 = (  Q 3  )(  R 3  )
  121C    3003                 MOVE.W D3,D0;     ! D0 = (  Q 3  )(  Q 2  )
  121E    4840                 SWAP   D0;        ! D0 = (  Q 2  )(  Q 3  )
  1220    4241                 CLR.W  D1;        ! D1 = (  R 3  )(   0   )
  1222    4841                 SWAP   D1;        ! D1 = (   0   )(  R 3  )
  1224    2C00                 MOVE.L D0,D6;     ! D6 = (  Q 2  )(  Q 3  )
  1226    2E01                 MOVE.L D1,D7;     ! D7 = (   0   )(  R 3  )
  1228    4E40                 TRAP   0;;        ! RUECKSPRUNG ZUM MONITOR

SYMBOL          BLOCK          ZEILE      LAENGE     WERT
-------------------------------------------------------------------------

V_NULL                           57                   $1218  PC-R.
```

Programmbeispiel 4

```
                          ! UMWANDLUNG EINER HEXADEZIMALEN ZAHL IN EINE
                          ! DEZIMALE ZAHL (MAX. DEZ. ZAHL = 99999)
                          !
                          ! DIE HEXADEZIMALE ZAHL STEHT IN D0
                          ! DIE DEZIMALE ZAHL STEHT NACH DER UMWANDLUNG IN D1
                          !
                          ! DEFINATOR
                          !
                          HEXZAHL: EQU.L D3;
                          DEZZAHL: EQU.L D1;
                          DEZMAX:  EQU.L D2;
                          !
                          ! INITIALISIERUNG
                          !
                           ORG $1020;          ! PROGRAMMSTARTADRESSE
                          !
 1020    2600                     MOVE.L D0,HEXZAHL;     ! D0 RETTEN
 1022    4281                     CLR    DEZZAHL;        ! ERGEBNISZELLE = 0
 1024    243C00002710             MOVE.L #10000,DEZMAX;  ! DURCH DEZMAX WIRD
                                                         !FESTGELEGT WIE
                                                         !GROSS DIE DEZIMALE
                                                         !ZAHL MAXIMAL SEIN
                                                         !DARF.
                                                         !-MAX. DEZ. ZAHL =
                                                         !(DEZMAX * 10) - 1
                                                         !-DEZMAX = 10 EXP N
                          !                              !-N DARF MAX. 4 SEIN
                          !
                          ! HAUPTPROGRAMM
                          !
 102A    86C2             UMWAND: DIVU   DEZMAX,HEXZAHL;! HEXZAHL:=
                                                         !HEXZAHL / DEZMAX
 102C    6108                     BSR.S  RESTTR;         AUFRUFEN DES UNTER-
                                                         !PROGRAMMES "REST
                                                         !TRENNEN"
 102E    0C420000                 CMPI.W 0,DEZMAX;       ! PRUEFEN OB DIE ZAH-
 1032    66F6                     BNE.S  UMWAND;         !LENUMWANDLUNG ABGE-
                                                         !SCHLOSSEN IST
 1034    4E40                     TRAP   0;;             ! RUECKSPRUNG ZUM
                                                         !MONITOR
                          !
                          !
                          ! UNTERPROGRAMM "REST TRENNEN":
                          ! DIESES UNTERPROGRAMM WIRD N+1 MAL DURCHLAUFEN
                          !
 1036    C2FC0010         RESTTR: MULU   16,DEZZAHL;       ! BISHER ERMITTELTE
                                                           !DEZ. ZAHL * 16
 103A    D243                     ADD.W  HEXZAHL,DEZZAHL;  ! ERGEBNIS AUS DER
                                                           !DIVISION (OHNE
                                                           !REST) + BERECH-
                                                           !NETER DEZ. ZAHL
 103C    4843                     SWAP   HEXZAHL;          ! ERGEBNIS AUS DER
 103E    0283 0000FFFF            ANDI.L $0000FFFF,HEXZAHL;!DEVISION VOM REST
                                                           !TRENNEN
 1044    84FC000A                 DIVU   10,DEZMAX;        ! N = N - 1
 1048    4E75                     RTS;                     ! RUECKSPRUNG ZUM
                                                           !HAUPPROGRAMM
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT
DEZMAX		15	LONG	D2
DEZZAHL		14	LONG	D1
HEXZAHL		13	LONG	D3
RESTTR		51		$1036 PC-R.
UMWAND		36		$102A PC-R.

Programmbeispiel 5

```
                                  ! UMWANDLUNG EINER DEZIMALEN ZAHL (MAX. 99999) IN
                                  !EINE HEXADEZIMALE ZAHL


                                  ! IN D0 STEHT DIE DEZIMALE ZAHL
                                  ! IN D1 STEHT DIE HEXADEZIMALE ZAHL

                                  !  DEFINATOR

                                  DEZZAHL:  EQU  D0;
                                  HEXZAHL:  EQU  D1;
                                  DEZWERT:  EQU  D4;
                                  MASKE:    EQU  $0000000F;


                                            ORG $1A00;

  1A00    2400                    MOVE.L DEZZAHL,D2;   ! DEZIMALZAHL RETTEN
  1A02    4281                    CLR.L  HEXZAHL;      ! ERGEBNISZELLE = 0
  1A04    283C00000001            MOVE.L #1,DEZWERT;   ! DEZWERT FUER DIE
                                                       !ERSTE STELLE

  1A0A    2602            NOCHMAL: MOVE.L D2,D3;       ! D2 RETTEN
  1A0C    02830000000F            ANDI.L #MASKE,D3;    ! AKTUELLE STELLE AUS-
                                                       !BLENDEN
  1A12    C6C4                    MULU   DEZWERT,D3;
  1A14    D283                    ADD.L  D3,HEXZAHL;   ! ERGEBNIS AUS DER MUL-
                                                       !TIPLIKATION + BISHER
                                                       !ERMITTELTER HEXZAHL
  1A16    E88A                    LSR.L  4,D2;         ! DEZIMALZAHL UM EINE
                                                       !STELLE VERSCHIEBEN
  1A18    6706                    BEQ.S  FERTIG;
  1A1A    C8FC000A                MULU   #10,DEZWERT;  ! DEZWERT FUER NAECHSTE
                                                       !STELLE BESTIMMEN
  1A1E    60EA                    BRA.S  NOCHMAL;      ! NAECHSTE STELLE BEAR-
                                                       !BEITEN

  1A20    4E40            FERTIG:  TRAP   0;;          ! RUECKSPRUNG ZUM
                                                       !MONITOR
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT
DEZWERT		14		D4
DEZZAHL		12		D0
FERTIG		40		$1A20 PC-R.
HEXZAHL		13		D1
MASKE		15		$F
NOCHMAL		25		$1A0A PC-R.

Programmbeispiel 6

```
                                  DER INHALT ZWEIER TABELLEN WIRD AUF
                                  GLEICHHEIT UEBERPRUEFT


                                  ! IN D0 STEHT DIE LAENGE DER TABELLEN (ANZAHL DER
                                  ! LANGWOERTER)

                                  ! IN A0 STEHT DIE ANFANGSADRESSE DER ERSTEN TABELLE

                                  ! IN A1 STEHT DIE ANFANGSADRESSE DER ZWEITEN TABELLE
```

Fortsetzung Programmbeispiel 6

```
                        ! WENN DER INHALT DER BEIDEN TABELLEN UEBEREINSTIMMT
                        ! IST D1 = 0

                        ! TRIFFT DIES NICHT ZU STEHT IN D1 DIE SPEICHER-
                        ! ADRESSE DER TABELLE 2 BEI DER DER FEHLER AUFTRAT

                        ! IN D2 (TABELLE 1) UND D3 (TABELLE 2) STEHEN DANN
                        ! DIE SPEICHERINHALTE DER BEIDEN TABELLEN


                        ! DEFINATOR

                        PROGRAMM:       EQU    $1100; ! PROGRAMMSTARTADRESSE
                        TABELLE1:       EQU    $2000; ! ANFANGSADRESSE
                        TABELLE2:       EQU    $3000; ! ANFANGSADRESSE
                        LAENGE:         EQU     D0;   ! ANZAHL DER LANGWOETER
                        FEHLERADRESSE:  EQU     D1;
                        ANZAHL:         EQU     D4;
                        GLEICH:         EQU.W $FFFF; ! ENTSPRICHT -1


                         ORG  PROGRAMM;

                        ! INITIALISIERUNG

  1100    2800                          MOVE.L LAENGE,ANZAHL;    ! INHALT VON
                                                                 !LAENGE RETTEN
  1102    207C00002000                  MOVE.L #TABELLE1,A0;
  1108    227C00003000                  MOVE.L #TABELLE2,A1;
  110E    003C0010                      ORI    #$10,CCR;         ! Z FLAGGE = 1
  1112    4281                          CLR.L  FEHLERADRESSE;
  1114    4282                          CLR.L  D2;
  1116    4283                          CLR.L  D3;
  1118    6002                          BRA.S  NEUER_WERT;


                        ! HAUPTPROGRAMM

  111A    B3S8          VERGLEICH:: CMPM.L (A0)+,(A1)+;
  111C    56CCFFFC      NEUER_WERT: DBNE   ANZAHL,VERGLEICH;! VERZWEIGEN,
                                                            !WENN INHALT
                                                            !GLEICH UND AN-
                                                            !ZAHL NICHT -1
  1120    0C840000FFFF                  CMPI.L #GLEICH,ANZAHL;   ! UEBERPRUEFEN
  1126    6706                          BEQ.S  FERTIG;           !OB ANZAHL = -1
  1128    2420                          MOVE.L -(A0),D2;
  112A    2621                          MOVE.L -(A1),D3;
  112C    2209                          MOVE.L A1,FEHLERADRESSE;
  112E    4E40          FERTIG:         TRAP   0;;               ! RUECKSPRUNG
                                                                 !ZUM MONITOR

SYMBOL            BLOCK            ZEILE      LAENGE     WERT
-----------------------------------------------------------------------------

ANZAHL                             32                    D4
FEHLERADRESSE                      31                    D1
FERTIG            VERGLEICH        63                    $112E   PC-R.
GLEICH                             33         WORD       $FFFF
LAENGE                             30                    D0
NEUER_WERT        VERGLEICH        54                    $111C   PC-R.
PROGRAMM                           27                    $1100
TABELLE1                           28                    $2000
TABELLE2                           29                    $3000
VERGLEICH         VERGLEICH        53                    $111A   PC-R.
```

Programmbeispiel 7

```
                        ! KOPIERPROGRAMM
                        ! ES WERDEN N  LANGWOERTER (32 BIT) AUS DEM ADRESS-
                        ! BEREICH 1 IN DEN ADRESSBEREICH 2 KOPIERT
                        !
                        ! DEFINATOR
                        !
                        ANZAHL: EQU $08;    !N = 8
                        QUELLE: EQU $2000;  !QUELLENZEIGER (ZEIGT AUF DIE
                                            !ANFANGSADRESSE DES ADRESSBE-
                                            !REICHS 1)
                        ZIEL:   EQU $3000;  !ZIELZEIGER (ZEIGT AUF DIE AN-
                                            !FANGSADRESSE DES ADRESSBE-
                                            !REICHS 2)
                        !
                        !
                        ! INITIALISIERUNG
                        !
                         ORG $1000;                 !PROGRAMMSTARTADRESSE
                        !
1000    303C0008                MOVE.W #ANZAHL,D0; !FESTLEGUNG WIEVIELE LANG-
                                                   !WOERTER KOPIERT WERDEN
1004    227C00002000            MOVE.L #QUELLE,A1; !SETZEN DES QUELLENZEIGERS
100A    247C00003000            MOVE.L #ZIEL,A2;   !SETZEN DES ZIELZEIGERS

                        ! KOPIERROUTINE
                        !
1010    04400001                SUBI.W 1,D0;       !D0=D0-1 SONST N=N+1
1014    24D9            KOPIE:  MOVE.L (A1)+,(A2)+;!KOPIEREN EINES LANGWORTES
                                                   !UND ZEIGER AUF DIE NAECHS
                                                   !TEN ADRESSEN STELLEN
1016    51C8FFFC                DBF    D0,KOPIE;   !KOPIERVORGANG BEENDET?
101A    4E75                    RTS;;              !ENDE
                        !
                        !
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT
ANZAHL		10		$8
KOPIE		31		$1014 PC-R.
QUELLE		11		$2000
ZIEL		14		$3000

Programmbeispiel 8

```
                                  ! EINE NEUE ZAHL (MAX. $FFFF) WIRD IN EINE
                                  !ZAHLENLISTE EINGEFUEGT

                                  ! DIE ZAHLEN SIND NACH IHRER GROESSE SORTIERT
                                  ! DIE GROESSTE ZAHL STEHT AM ENDE DER ZAHLENLISTE

                                  ! IN A0 STEHT DIE ANFANGSADRESSE DER ZAHLENLISTE
                                  ! IN A1 STEHT DIE AKTUELLE ENDADRESSE DER ZAHLEN-
                                  !LISTE
                                  ! IN D0 STEHT DIE NEUE ZAHL


                                  ANFADR:       EQU   A0; ! ANFANGSADRESSE DER ZAHLEN-
                                                          !LISTE
                                  ENDADR:       EQU   A1; ! AKTUELLE ENDADRESSE DER ZAH-
                                                          !LENLISTE
                                  SUCH_ENDADR:  EQU   A2;
                                  ZEIGER:       EQU   A3;
                                  NEUE_ZAHL:    EQU   D0:


                                   ORG $1500;

  1500    3448                          MOVE.W ANFADR,SUCH_ENDADR; ! ANFANGSADRESSE
                                                                   !RETTEN
  1502    95FC00000001                  SUBA   #1,SUCH_ENDADR;
  1508    3649                          MOVE.W ENDADR,ZEIGER;
  150A    B059                          CMP.W  (ENDADR)+,NEUE_ZAHL;! UEBERPRUEFEN OB
  150C    640C                          BCC.S  ABSPEICH;           !NEUE ZAHL ) MAX.
                                                                   !ZAHL & ENDADRES-
                                                                   !SE + 1
  150E    37530002          SCHIEBEN:   MOVE.W (ZEIGER),2(ZEIGER); ! UMSPEICHERN
  1512    B063                          CMP.W  -(ZEIGER),NEUE_ZAHL;! UEBERPRUEFEN
  1514    6404                          BCC.S  ABSPEICH;           !OB NEUE ZAHL )
                                                                   !SPEICHERINHALT &
                                                                   !ZEIGER - 1
  1516    B6CA                          CMP.W  SUCH_ENDADR,ZEIGER; ! UEBERPRUEFEN OB
  1518    66F4                          BNE.S  SCHIEBEN;           !ZAHLENLISTENAN-
                                                                   !FANG ERREICHT
  151A    37400002          ABSPEICH:   MOVE.W NEUE_ZAHL,2(ZEIGER);! NEUE ZAHL AB-
                                                                   !SPEICHERN
  151E    4E40                          TRAP   0;;                 ! RUECKSPRUNG ZUM
                                                                   !MONITOR
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT	
ABSPEICH		42		$151A	PC-R.
ANFADR		15		A0	
ENDADR		17		A1	
NEUE_ZAHL		21		D0	
SCHIEBEN		34		$150E	PC-R.
SUCH_ENDADR		19		A2	
ZEIGER		20		A3	

Programmbeispiel 9

```
                        ! EIN DATENBLOCK WIRD SO SORTIERT, DASS DAS WORT MIT
                        !DEM GROESSTEN WERT AN DER NIEDRIGSTEN SPEICHERADRES-
                        !SE ABGELEGT IST.

                        ! IN A0 STEHT DIE ANFANGSADRESSE DES SPEICHERBEREICHS
                        ! IN A1 STEHT DIE ENDADRESSE DES SPEICHERBEREICHS


                        ANFANG:  EQU  $2000;
                        ENDE:    EQU  $3000;

                                 ORG $1C00;

1C00    227C00003000             MOVE.L ENDE,A1;
1C06    247C00002000             MOVE.L ANFANG,A2;   ! ANFANGSADRESSE RETTEN
1C0C    204A            START:   MOVE.L A2,A0;       ! ANFANGSADRESSE LADEN
1C0E    B148            NAECHSTE: CMPM.W (A0)+,(A0)+;! WENN ZWEITER WERT )
                                                     !ALS ERSTER WERT DANN
1C10    6208                     BHI.S  TAUSCHEN;    !TAUSCHEN
1C12    4AE0                     TST.W  -(A0);       ! DUMMY FUER DEKREMENT
1C14    B3C8                     CMP.L  A0,A1;       ! TESTEN OB ENDE ER-
1C16    66F6                     BNE.S  NAECHSTE;    !REICHT
1C18    4E40                     TRAP   0;

1C1A    3020            TAUSCHEN: MOVE.W -(A0),D0;   ! ZWEITER WERT NACH D0
1C1C    3220                     MOVE.W -(A0),D1;    ! ERSTER WERT NACH D1
1C1E    48900003                 MOVEM.W D0/D1,(A0);! UMGEKEHRT ABSPEICHERN
1C22    60E8                     BRA.S  START;;      ! NEUER VERGLEICH
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT	
ANFANG		11		$2000	
ENDE		12		$3000	
NAECHSTE		19		$1C0E	PC-R.
START		18		$1C0C	PC-R.
TAUSCHEN		27		$1C1A	PC-R.

Programmbeispiel 10

```
                    ! EINE SPEICHERZELLE (16 BIT) WIRD UNTERSUCHT OB
                    !DEREN INHALT GLEICH BLEIBT, WENN MAN DIE ZELLE VON
                    !LINKS ODER VON RECHTS LIEST

                      TRIFFT DIES ZU STEHT IN D1 000000FF SONST 00000000

                      IN D0 STEHT DIE ZU UNTERSUCHENDE SPEICHERZELLE


                    SPEICHERZELLE: EQU  D0;
                    ERGEBNIS:      EQU  D1;
                    ANZAHL:        EQU  D4;

                              ORG $1900;

  1900    2400                MOVE.L  SPEICHERZELLE,D2;
  1902    7807                MOVEQ.L #7,ANZAHL;
  1904    4281                CLR.L   ERGEBNIS;
  1906    E24A        NOCHMAL: LSR.W  1,D2;
  1908    E353                ROXL.W  1,D3;
  190A    51CCFFFA            DBRA    ANZAHL,NOCHMAL;
  190E    B602                CMP.B   D2,D3;
  1910    57C1                SEQ.W   ERGEBNIS;
  1912    4E40                TRAP    0;;
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT
ANZAHL		14		D4
ERGEBNIS		13		D1
NOCHMAL		21		$1906 PC-R.
SPEICHERZELLE		12		D0

Programmbeispiel 11

```
                        ! ZEICHENZUORDNUNGSPROGRAMM

                        ! IN D0 STEHT EINE HEX ZAHL (0 - $F)

                        ! IN D1 STEHT DANN DIE ASCII ZAHL
                        ! IN D2 STEHT DANN DIE BCD ZAHL
                        ! IN D3 STEHT DANN DIE GRAY CODE ZAHL

                         ORG $1000;              ! PROGRAMMSTARTADRESSE

1000    4281             CLR.L  D1;
1002    4282             CLR.L  D2;
1004    4283             CLR.L  D3;
1006    41FAF8F8         LEA    ASCII_TABLE,A0;! ZEIGER STEHT AUF DER ERSTEN
                                                !ADRESSE DER ASCII TABELLE
100A    12300800         MOVE.B 0(A0,D0),D1;    ! DAS ENTSPRECHENDE ASCII ZEI-
                                                !CHEN WIRD IN D1 ABGESPEICHERT
100E    14300810         MOVE.B 16(A0,D0),D2;   ! DAS ENTSPRECHENDE BCD ZEI-
                                                !CHEN WIRD IN D2 ABGESPEICHERT
1012    16300820         MOVE.B 32(A0,D0),D3;   ! DAS ENTSPRECHENDE GRAY CODE
                                                !ZEICHEN WIRD IN D3 ABGESP.
1016    4E40             TRAP   0;              ! RUECKSPRUNG ZUM MONITOR

                         ORG $900;
                        ASCII_TABLE:
 900    30                DC.B $30;
 901    31                DC.B $31;
 902    32                DC.B $32;
 903    33                DC.B $33;
 904    34                DC.B $34;
 905    35                DC.B $35;
 906    36                DC.B $36;
 907    37                DC.B $37;
 908    38                DC.B $38;
 909    39                DC.B $39;
 90A    41                DC.B $41;
 90B    42                DC.B $42;
 90C    43                DC.B $43;
 90D    44                DC.B $44;
 90E    45                DC.B $45;
 90F    46                DC.B $46;

                        BCD_TABELLE:
 910    00                DC.B   0;
 911    01                DC.B   1;
 912    02                DC.B   2;
 913    03                DC.B   3;
 914    04                DC.B   4;
 915    05                DC.B   5;
 916    06                DC.B   6;
 917    07                DC.B   7;
 918    08                DC.B   8;
 919    09                DC.B   9;
 91A    10                DC.B $10;
 91B    11                DC.B $11;
 91C    12                DC.B $12;
 91D    13                DC.B $13;
 91E    14                DC.B $14;
 91F    15                DC.B $15;
```

Fortsetzung Programmbeispiel 11

```
                                GRAY_CODE_TABELLE:
   920     00                     DC.B    0;
   921     01                     DC.B    1;
   922     03                     DC.B    3;
   923     02                     DC.B    2;
   924     06                     DC.B    6;
   925     07                     DC.B    7;
   926     05                     DC.B    5;
   927     04                     DC.B    4;
   928     0C                     DC.B   $C;
   929     0D                     DC.B   $D;
   92A     0F                     DC.B   $F;
   92B     0E                     DC.B   $E;
   92C     0A                     DC.B   $A;
   92D     0B                     DC.B   $B;
   92E     09                     DC.B    9;
   92F     08                     DC.B    8;

SYMBOL            BLOCK            ZEILE      LAENGE     WERT
-------------------------------------------------------------------------------

ASCII_TABLE                          28        BYTE       $900  PC-R.
BCD_TABELLE                          46        BYTE       $910  PC-R.
GRAY_CODE_TABE                       64        BYTE       $920  PC-R.
```

Programmbeispiel 12

```
                    ! RAMTESTPROGRAMM (DER SPEICHERINHALT DES RAMS WIRD
                    !NICHT VERAENDERT)

                    ! IN A0 STEHT DIE ERSTE RAMTESTADRESSE
                    ! IN A1 STEHT DIE LETZTE RAMTESTADRESSE+1


                    ! WURDE KEIN FEHLER FESTGESTELLT IST DIE
                    !Z-FLAGGE = 1
                    ! IN A2 STEHT DIE ADRESSE 0000 0000

                    ! WURDE EIN FEHLER FESTGESTELLT IST DIE
                    !Z-FLAGGE = 0
                    ! IN A2 STEHT DIE ADRESSE BEI DER DER
                    !FEHLER AUFTRAT (UMSPEICHERUNGSFEHLER)
                    ! IN A2 STEHT DIE ERSTE ODER LETZTE
                    !RAMTESTADRESSE+1 (TESTSUMMENFEHLER)

                    ! DEFINATOR

                    RAMANF:     EQU.L  A0;
                    RAMEND:     EQU.L  A1;
                    ZEIGER:     EQU.L  A2;
                    EXORZELLE:  EQU.L  D0;
                    RAMDATEN:   EQU.L  D1;
                    TESTSUMM:   EQU.L  D2;
                    FEHLERADR:  EQU.L  D3;


                     ORG $900;

                    ! INITIALISIERUNG

900     2448                MOVE.L RAMANF,ZEIGER;
902     4283                CLR.L  FEHLERADR;
904     4240                CLR.W  EXORZELLE;

                    ! TESTSUMME BESTIMMEN

906     321A        TESTSUM: MOVE.W (ZEIGER)+,RAMDATEN;
908     B340                EOR.W  RAMDATEN,EXORZELLE;
90A     B5C9                CMP.L  RAMEND,ZEIGER;
90C     66F8                BNE.S  TESTSUM;
90E     3400                MOVE.W EXORZELLE,TESTSUMM;

                    ! TESTSCHLEIFE 1

910     2448                MOVE.L RAMANF,ZEIGER;
912     4240                CLR.W  EXORZELLE;
914     3212        TEST_1:  MOVE.W (ZEIGER),RAMDATEN;
916     B340                EOR.W  RAMDATEN,EXORZELLE;! TESTSUMME ERZEU-
                                                      !GEN
918     4641                NOT.W  RAMDATEN;          ! NEUE DATEN
91A     3481                MOVE.W RAMDATEN,(ZEIGER); !ABSPEICHERN
91C     B25A                CMP.W  (ZEIGER)+,RAMDATEN;! ABSPEICHERUNG
91E     6706                BEQ.S  DURCHG1;           !UEBERPRUEFEN
920     4A83                TST.L  FEHLERADR;         ! TESTEN OB SCHON
922     6602                BNE.S  DURCHG1;           !EIN FEHLER FEST-
                                                      !GESTELLT WORDEN
                                                      !IST
924     260A                MOVE.L ZEIGER,FEHLERADR;  ! FEHLERADRESSE
                                                      !ABSPEICHERN
926     B5C9        DURCHG1: CMP.L  RAMEND,ZEIGER;
928     66EA                BNE.S  TEST_1;
```

Fortsetzung Programmbeispiel 12

```
92A   B440                    CMP.W  EXORZELLE,TESTSUMM;! TESTSUMME
92C   6706                    BEQ.S  TSTEND1;           !UEBERPRUEFEN
92E   4A83                    TST.L  FEHLERADR;         ! TESTEN OB SCHON
930   6602                    BNE.S  TSTEND1;           !EIN FEHLER FEST-
                                                        !GESTELLT WORDEN
                                                        !IST
932   260A                    MOVE.L ZEIGER,FEHLERADR;  ! RAMENDADRESSE+1
                                                        !ABSPEICHERN

                        ! TESTSCHLEIFE 2

934   4240            TSTEND1: CLR.W  EXORZELLE;
936   3222            TEST_2:  MOVE.W -(ZEIGER),RAMDATEN;
938   4641                    NOT.W  RAMDATEN;          ! URSPRUENGLICHE
                                                        !DATEN
93A   B340                    EOR.W  RAMDATEN,EXORZELLE;! TESTSUMME ER-
                                                        !ZEUGEN
93C   3481                    MOVE.W RAMDATEN,(ZEIGER); ! DATEN ABSPEICH.
93E   B252                    CMP.W  (ZEIGER),RAMDATEN; ! ABSPEICHERUNG
940   6706                    BEQ.S  DURCHG2;           !UEBERPRUEFEN
942   4A83                    TST.L  FEHLERADR;         ! TESTEN OB SCHON
944   6602                    BNE.S  DURCHG2;           !EIN FEHLER FEST-
                                                        !GESTELLT WORDEN
                                                        !IST
946   260A                    MOVE.L ZEIGER,FEHLERADR;  ! FEHLERADRESSE
                                                        !ABSPEICHERN

948   B5C8            DURCHG2: CMP.L  RAMANF,ZEIGER;
94A   66EA                    BNE.S  TEST_2;

94C   B440                    CMP.W  EXORZELLE,TESTSUMM;! TESTSUMME
94E   6706                    BEQ.S  TSTEND2;           !UEBERPRUEFEN
950   4A83                    TST.L  FEHLERADR;         ! TESTEN OB SCHON
952   6602                    BNE.S  TSTEND2;           !EIN FEHLER FEST-
                                                        !GESTELLT WORDEN
                                                        !IST
954   260A                    MOVE.L ZEIGER,FEHLERADR;  ! ANFANGSADRESSE
                                                        !ABSPEICHERN

956   2443            TSTEND2: MOVE.L FEHLERADR,ZEIGER; ! ZEIGER AUF FEH-
                                                        !LERADRESSE STEL-
                                                        !LEN
958   4A83                    TST.L  FEHLERADR;         ! Z-FLAGGE BEEIN-
                                                        !FLUSSEN
95A   4E40                    TRAP   0;;
```

SYMBOL	BLOCK	ZEILE	LAENGE	WERT	
DURCHG1		65		$926	PC-R.
DURCHG2		95		$948	PC-R.
EXORZELLE		26	LONG	D0	
FEHLERADR		29	LONG	D3	
RAMANF		23	LONG	A0	
RAMDATEN		27	LONG	D1	
RAMEND		24	LONG	A1	
TESTSUM		42		$906	PC-R.
TESTSUMM		28	LONG	D2	
TEST_1		52		$914	PC-R.
TEST_2		80		$936	PC-R.
TSTEND1		79		$934	PC-R.
TSTEND2		107		$956	PC-R.
ZEIGER		25	LONG	A2	

Aus unserer Buchreihe

Der 16bit-Mikroprozessor SC 68000

Eigenschaften

Bearbeitet von J. Koch, 105 Seiten, 43 Abbildungen, 1982, DM 13,–

ISBN 3-87095-255-5

Der SC 68000 ist ein außerordentlich leistungsfähiger 16bit-Mikroprozessor, dessen interne Registerwortbreite 32 bit beträgt. Der Befehlsvorrat umfaßt 56 leistungsfähige Grundbefehle, die 14 Adressierungsarten verwenden und 5 Datentypen verarbeiten können.
Das Buch enthält eine ausführliche Beschreibung der Rechnerarchitektur und der externen Schnittstelle des SC 68000. Die verschiedenen Bus-Operationen werden im einzelnen dargestellt und die Verarbeitungszustände (privilegierte Zustände, Ausnahmeverarbeitung) detailliert beschrieben. Der SC 68000 verfügt über einen asynchronen Bus, kann aber auch mit synchronen 8bit-Peripherieschaltungen zusammenarbeiten. Außerdem beinhaltet das Buch die wichtigsten Informationen über den Befehlsvorrat sowie die Ausführungszeiten (Taktperioden) der einzelnen Befehle.

Zu beziehen durch den Buchhandel
Verlag BOYSEN+MAASCH,
Hermannstr. 31, 2000 Hamburg 1